FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Automated Scalable Platform for Packet Traffic Analysis

Miguel José Cavadas Santos

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Scientific supervision by:
Ricardo Santos Morla

Aditional supervision by:
Jorge Manuel Gomes Barbosa

February 23, 2016

Abstract

In these days a growing rate of cybercrime exists, in both number and complexity of the attacks that are being made. This crime is becoming more powerful, persistent, costly and focused on targets by the day. Preventing and detecting this dangerous situation is key to protect any Internet based company, and, because of this, security best practices are followed, firewalls are implemented, and very powerful Intrusion Detection Systems and Intrusion Prevention Systems are developed. All types of IDS have their particular strengths but not a single one is perfect: they can fail to detect some specially crafted attacks, not have enough processing speed to deal with the threat or even assume a certain connection to be an attack when it is not.

There are two major types of IDS - signature and anomaly based - and much work is constantly being done to enhance them. In this thesis we studied Snort, a very popular free signature-based IDS. Most of the state of the art solutions found for Snort address issues like rule inefficiency, improving detection capability, using big data analytics on the alerts, and studying the false positive ratio. No work was found that tackles the performance issue when too many rules are loaded into Snort, or as it is also called, the "Security over Connectivity" problem. There is also a lack of studies about the integration of Snort processing with MapReduce jobs.

With that in mind, the goal of this thesis is to study the impact of distributing the processing of Snort through a computer cluster and compare the results obtained, such as times measured and alerts produced, with the state of the art solutions. To accomplish this we developed The Hunter, a completely automated and scalable platform that distributes packet captures for Snort to process in an already deployed computer cluster, while also allowing integration with Hadoop and MapReduce.

The results obtained with the developed tool confirmed that a distributed approach to Snort can indeed coexist with Hadoop in its core, while maintaining a fast processing time and good detection rate.

Acknowledgements

This dissertation thesis marks the end of my academic journey. As such, there were many people who influenced and helped me grow through this five and a half years.

A special thank you to Ricardo Morla, the supervisor of this thesis, who allowed me to follow an informatic security theme, which was one of the subjects I enjoyed more when studying, and guided me throughout the entire project.

To my parents, who believed in me and gave me wings to follow my dreams.

To all the friends I made during this time, and to the ones I already had: a big thank you!

"When everything seems to be going against you, remember that the airplane takes off against the wind, not with it."

Henry Ford

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

API Application Programming Interface

ARP Address Resolution Protocol

DNS Domain Name Server

DoS Denial Of Service

FTP File Transfer Protocol

HDFS Hadoop Distributed File System

HIDS Host-based Intrusion Detection System

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IoT Internet of Things

IP Internet Protocol

IPS Intrusion Prevention System

KNN K-Nearest Neighbor

LAN Local Area Network

NIDS Network-based Intrusion Detection Service

OS Operating System

PCA Principal Component Analysis

RDF Radial Basis Function

ROC Receiver Operating Characteristic

SMS Short Message System

SQL Structured Query Language

SSH Secure Shell

SSL Secure Socket Layer

SVM Support Vector Machine

TCP Transmission Control Protocol

TLS Transport Layer Security

TTL Time To Live

UDP User Datagram Protocol

URL Uniform Resource Locator

WPA Wi-Fi Protected Access

WPA2 Wi-Fi Protected Access 2

XSS Cross Site Scripting

YARN Yet Another Resource Negotiator

Chapter 1

Introduction

1.1 Context

Lead by the constant improvements of devices and the easiness of Internet access we are currently living the era of the Internet of Things (IoT). This means that the computing, processing and connectivity that we were used to when using desktop computers is now being transferred to all kinds of devices, like smartphones, tablets, security cameras, coffee machines, shoes that connect to the social media and cat feeders that release food with distant online commands. This is not a new movement, but it is nonetheless a trending one with great impacts to the way people are used to live.

With the rise of the IoT everyone is more connected than ever before. With every day that passes, more and more different types of devices get online, and this number just keeps growing. According to Gartner in 2016 there should be 6.4 billion devices connected to the Internet, which corresponds to a 30% growth when looking at 2015, and that number can reach 20.8 billion units by 2020 [1].

There is great benefit in connecting sensors, smartphones and everyday devices together in the IoT, usually with the intent to make the lives of people easier and to enhance their way of life. For instance if all devices in a household could communicate with each other they could potentially learn when they are being needed, or not, and save money in energy bills. The health sector could benefit greatly by making different devices that people usually carry, like smartphones and watches, constantly monitoring them, and automatically warn authorities if anything wrong happens, like abnormal heart rate or respiratory failure.

However, every single one of this variety of devices has one problem in common: just by being connected to the Internet, they are all vulnerable. Hackers everywhere are now focusing their attention to this new connected devices. For instance, the smartphone can do most things a personal computer can: contact friends, access social media and perform bank activities, while at the same time being, in most cases, less secure, mainly because most people do not update their devices regularly and do not use any kind of virus or malware protection, like they do on their personal computers. According to a Kaspersky Lab and Interpol report, Android phones are the

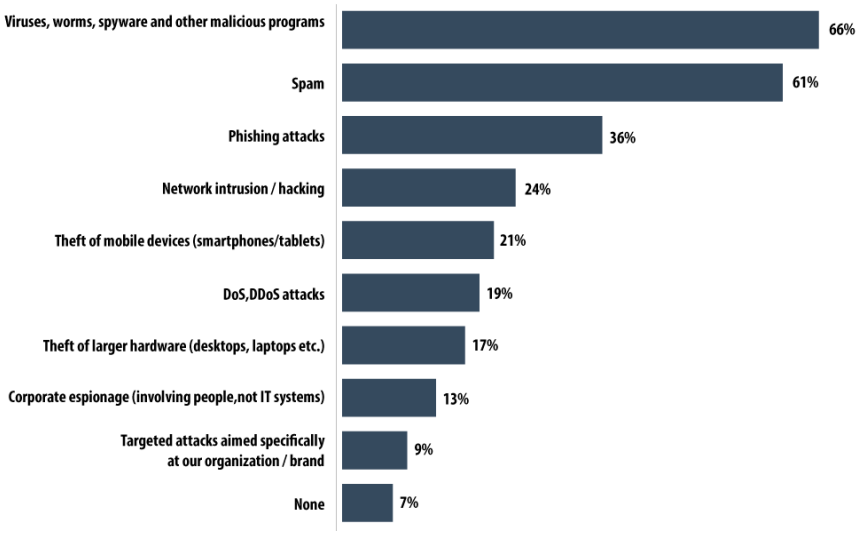


Figure 1.1: Different attacks and respective proportion [3]

more targeted, due to the easiness of making and tricking people into installing malware through apps with unverified sources [2].

Successfully exploiting an insecure device can lead to a breach in a network, with disastrous consequences. If a company invested in security for the protection of their online assets and someone can exploit a newly connected coffee machine, there is a chance to pivot to other devices on the intranet because the attacker is already inside. With this rise of the Internet of Things, people and companies should take cyber security even more serious and make sure to apply measures to protect themselves.

This is why security research is so important nowadays, and any company that has private assets online should follow best practices for network security. This includes setting up firewalls, protecting virtual Local Area Network (LAN) and deploying Intrusion Detection System (IDS) and Intrusion Prevention System (IPS). Still, the efforts are not enough and companies are getting attacked each day. A survey by Kaspersky concluded that 91% of the organizations polled was victim to cyber attacks [3], and the proportion for different types of attacks are shown in figure 1.1.

Because of the constant cyber attacks, and their variety, much work is constantly being done in the information security field. Intrusion detection systems are being subjected to much research to enhance their detection capabilities. In this dissertation thesis, Snort, one of the most used IDS was studied.

1.2 Motivation

There are several different kinds of cyber protection anyone can implement, and IDS are just one of them. They can work with signatures, like Snort, where it is possible to find configuration files with written rules so that the IDS can compare the network traffic to find intrusions and output

alerts. Statistical anomaly based IDS have also seen much implementation and they make use of data mining algorithms to predict if a packet is an attack or not.

Both these techniques have their benefits but also have very distinct problems. Anomaly based IDS can detect outliers in traffic, for instance a newly discovered attack that was not been studied before. However they need a very good trained model to not output too many false positives. On the other hand signature based IDS have very good detection rate but they can not possible detect an attack that isn't part of the rules files.

Some IDS implementations are employed only in one host. Network packet capture can be done by this host or sent to it by a cluster of computers, but the processing done by the IDS to evaluate the integrity of the packets is only as good as the computer doing the job. This makes for a single point of failure that can potentially lead to a breach in the network, making it vulnerable against many cyber attacks. A Big Data approach to this problem can lead to a distributed IDS with much more processing capability and storage capacity, which is the aim of this dissertation thesis.

With the growing rate of the volume of data that is produced and stored it became trending to speak of Big Data, a term used to refer to very large data sets that couldn't be processed the usual way due to both time consumed and volume used. This new approach allowed companies to expand even further their knowledge of the huge amount of information they have stored, for instance the health care companies that use IBM's big data and analytics platform registered a 20% decrease in patient mortality by analyzing streaming patient data [4].

There are several different frameworks that work in big data, the most used being Hadoop, a java open source Apache platform for reliable, scalable, distributed computing [5] that answers the problem of how to efficiently handle very large data sets. With all its modules, Hadoop can single handedly distribute the data in a high-throughput access cluster and create jobs to efficiently query all information in record time, when compared to the same query without distributed computing [6].

In order to bring the best of both worlds, a tool that would distribute the processing capability of Snort through a Hadoop computer cluster was developed in this thesis.

1.3 Objectives

The main goal of this thesis is to study if the distribution of Snort can be a good contribution to the IDS community, and in what scenarios it can be more beneficial than the State of the Art solutions. Working with Hadoop is also mandatory, as this is one of the most used frameworks for big data analysis and distributed computing.

The main objectives are as follows:

- Perform studies on the way Snort usually functions and how it performs intrusion detection.
- Enhance the slow performance of Snort when using a large amount of rules to process packet captures by distributing its performance to a computer cluster.

- Develop a tool that can be completely automated and scalable, with the main objective of distributing Snort to a cluster, while allowing integration with the Hadoop framework.
- Test the developed solution in its time performance and accuracy detection.

1.4 Structure

The rest of this document is organized as follows. The State of the Art is presented in Chapter 2, where IDS and different solutions that address enhancement for them are explained. Recent work with distributed computing when applying the Hadoop framework for scalability and availability is also referenced. Chapter 3 depicts the proposed solution, a completely automated and scalable platform for distributing Snort processing. For testing IDS cyber attacks are needed, as such, the attacks performed to test the developed tool are detailed in Chapter 4. The results obtained and tests performed are explained on Chapter 5 and finally a conclusion to this thesis is made on Chapter 6.

Chapter 2

State of the Art

In this chapter a simplistic view on solutions done so far to enhance the effectiveness of IDS is presented, varying from different types of detection to different algorithms used, and uses of big data and Hadoop for cyber security.

Software and research in the field of network monitoring and packet analysis, which will be important for this dissertation goals in presented in section 2.1. Improved results for data mining algorithms used for intrusion detection, all tested with the famous KDD99 dataset, are shown in section 2.2. A brief explanation on IDS is given in section 2.3. Some recent improvements done in the field of distributed computing, and some uses for intrusion detection, are explained in section 2.4. Finally a discussion of the chapter is given in section 2.5.

2.1 Network Monitoring and Packet Logging

Since the beginning of LANs, network administrators had the need to capture and monitor the traffic that passed through their network. For this purpose exists a vast range of software that can successfully monitor a network and/or log the connections. Some of the most common tools for monitoring are presented next, and they may come with a graphical interface, may be open-source and may have unique extra features, but they all can serve as a backbone to a intrusion detection system.

2.1.1 TCPDump and Libpcap

Libpcap [7] is a unix based library that allows complete read access to all the packets in the network, even those not destined for the machine. It is a mandatory install if any monitoring or packet sniffer tool is to be installed in a Linux system.

TCPDump is a libpcap based packet analyzer [8]. It does not have a graphical interface, instead all information is shown on the command line. When a capture is started, TCPDump prints information about the packets that are going through a predetermined network interface. When using the verbose flag (-v) TCPDump displays the packet timestamp; several layer three Internet Protocol (IP) headers like flags, offset and Time To Live (TTL); source and destination

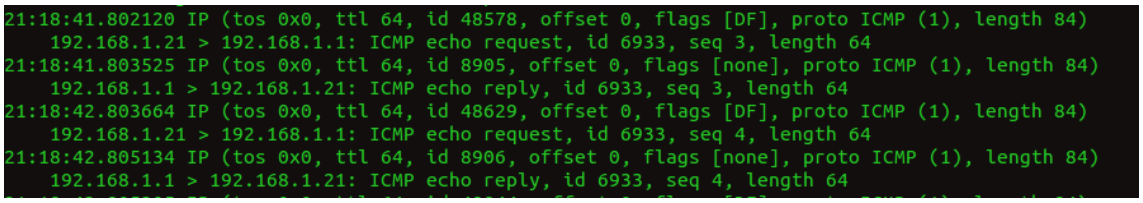


Figure 2.1: A ping captured with TCPDump

IP address and port used; about the Transmission Control Protocol (TCP) it shows what flags are active, the checksum, both acknowledgement and sequence number, the window size and packet length. The capture of a simple ping command can be viewed in figure 2.1.

Some of TCPDump major features are printing the information in ascii or hexadecimal, recording and reading the packets, limit for the size of the destination file and number of packets and even filters for protocol used, port and IP address are available.

2.1.2 Wireshark

Any student that starts to learn about networking, is also going to learn about Wireshark. This is one of the most used protocol analyzers [9], and considered one of the most important open-source applications of all time [10].

Wireshark has many different features and a great community updating it as necessary. It is mainly used for the ability to perform deep inspection of hundreds of different protocols in a simple graphical interface. The packets can be recorded to different formats, like TCPDump's libpcap, and can be read from Ethernet, Bluetooth, Frame Relay and others. It features a very strong filter option allowing great precision in finding a packet in the middle of all capture data, and can even support decryption for the most used protocols in cryptography such as Secure Socket Layer (SSL)/Transport Layer Security (TLS),Wi-Fi Protected Access (WPA) and Wi-Fi Protected Access 2 (WPA2) among others. It is possible to see two ping echo request packets and corresponding replies using Wireshark in figure 2.2.

2.1.3 SmartSniff, WinPcap and WinDump

While networking monitoring tools are usually used in Linux machines, Windows based systems can still be of assistance in this field. WinPcap is the industry standard library used in windows machines for capturing traffic in network interfaces [11]. It supports packet filtering at the kernel level and it also provides network statistics. For capturing traffic in any Windows machine, WinPcap is almost always a must have, for either open source and commercial packet loggers. With WinPcap one also as access to WinDump, which is, like the name suggests, a TCPDump version for Windows.

SmartSniff is a Wireshark alternative for Windows running machines [12]. It is a free software, made by NirSoft, which captures TCP/IP packets and, through a user interface, display the

```
37 1.541058000 192.168.1.21   192.168.1.1    ICMP   98 Echo (ping) request  id=0x1acb, seq=1/256, ttl=64 (reply in 38)
38 1.542523000 192.168.1.1    192.168.1.21   ICMP   98 Echo (ping) reply    id=0x1acb, seq=1/256, ttl=64 (request in 37)
39 2.542723000 192.168.1.21   192.168.1.1    ICMP   98 Echo (ping) request  id=0x1acb, seq=2/512, ttl=64 (reply in 40)
40 2.544202000 192.168.1.1    192.168.1.21   ICMP   98 Echo (ping) reply    id=0x1acb, seq=2/512, ttl=64 (request in 39)
```

Figure 2.2: A ping captured with Wireshark

recorded information in ascii or hexadecimal. It supports all Internet protocols and can also capture the traffic in different ways: through raw sockets, with a Microsoft Network Monitor Driver (which usually needs installing), and finally with WinPcap being the most advised way.

2.1.4 Network Monitoring Tools

There exists a variety of network monitoring tools that are widely used, even though they might not log and sniff Internet traffic, they are aimed at helping a network administrator understand exactly what is going on in his infrastructure and where might a problem be located.

Nagios is a famous open source network monitoring tool with over one million users worldwide [13]. After the user defines what services or components in the network are to be monitored, Nagios will provide all kinds of statistics and, in cause of failure, it will produce alerts for the network manager team, that can even be sent through Short Message System (SMS), allowing the team to quickly find and fix the alerted problem. Nagios can monitor any device in a network, web servers, databases, cloud computing, virtualization software, operating systems and many protocols. It also comes with a very user friendly web interface, where one can see all collected information in simple graphs. It is also very scalable, as it can monitor thousands of nodes. Because of the large community behind this open source project, plugins are very common and increase the range of devices and protocols this tool can monitor.

Zenoss is also a famous network management tool, free and open source [14]. It offers a single unified view of a network, its devices and nodes while controlling the availability and performance of all monitored events. It also has implemented complete support with Big Data infrastructures like Hadoop in order to completely satisfy large companies demands and give them proper high fidelity monitoring while reducing the overhead caused by all management operations.

Ntop is a free software that specializes in showing the user the statistic of the network usage in a web interface [15]. It supports libpcap for packet capture and it probes the network with nProbe, a netflow capturer, who can analyse Gbit networks with very few packet losses, while acting on the network's border gateway. Ntop also supports most layer 7 protocols analysis.

2.2 KDD99 Dataset

Until 2009 the Knowledge Discovery in Databases (KDD) Archive was able to maintain an online repository with large data sets. Their main goal was to help researchers discover new ways to scale data analysis algorithms to very large and complex data sets. In 1999 they published a database prepared by Lincoln Labs in order to motivate research in intrusion detection. This data set was based on raw TCPDump recorded for nine weeks inside a LAN simulating the behavior

that is expected of a military Air Force Base. After processing all the information obtained the dataset ended up with about five million connections, where a connection is "a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol" [16]. Each connection can be classified as normal or an attack, and there are four major types of attacks: Denial Of Service (DoS), unauthorized access from a remote machine, unauthorized access to local superuser privileges and surveillance and probing.

To this day, this dataset is still used to access the behavior of both new and variations of algorithms in data mining for intrusion detection, mainly because the theory for the methods of attack haven't changed, only the attacks themselves, and it is easy to compare the results obtained between different algorithms, for the dataset remains unchanged. Some recent work based on this KDD99 dataset that shows improved results in detecting intrusion will be presented next.

2.2.1 Classifying Network Traffic

Classification is a much studied supervised data mining problem that focus on identifying which subset of the population a observation belongs, taking into account the attributes of that particular observation. Usually a classification model is made based on a training set, corresponding to roughly 30% of the entries on the database, and then evaluated in the remaining 70% testing set. Some common algorithms used for classification are Decision Trees, K-Nearest Neighbor (KNN), Support Vector Machine (SVM) and the Naive Bayes classifier.

2.2.1.1 Decision Trees

Decision trees and other classifiers have been subjected to extended research in intrusion detection, because of their precision and speed and they fit very well in the KDD99 set due to all the entries being already defined as either an attack, and which one, or a normal connection. Different classification algorithms are normally compared by their predictive accuracy, speed in which they complete the learning model, scalability to very large datasets, robustness to noise and if the knowledge learned is easily interpreted. Decision Trees work by first using a specific decision tree algorithm, like ID3 or CART, to learn a model for the training set. At the start of this cycle, all the training observations are at the root of the decision tree, and recursively partitions are made based on selected attributes until a condition for stopping is achieved. Pruning can also be done in the middle of cycles to eliminate branches that reflect noisy data.

In [17] the researchers used the C4.5 algorithm, an improvement of ID3, with and without pruning, to find out if it could be used in intrusion detection successfully by learning the model with a version of the KDD99 Test Data, containing 311 073 connections, and evaluating on 10% of the KDD99 dataset, composed by 494 064 connections. They were able to achieve a 92.81% accuracy with C4.5 Decision Tree and an even better 95.09% accuracy with C4.5 Decision Tree with pruning which is a good result for the KDD99 dataset. Real-time capture and analysis of

network packets is mentioned as future work in the paper, which is also one objective of this dissertation.

2.2.1.2 Support Vector Machines

Support vector machines are also subject of extensive research and can be used for both classification and regression problems. SVM first maps the entries of the training set to a higher dimension and then calculates one or more hyperplanes, using a linear approach, that separates data from different classes. Kernels are functions used to help find and map the hyperplanes and are one key part of the success to SVMs. An optimized kernel solution was proposed in [18], and consisted of firstly reducing the data of the KDD99 dataset with Principal Component Analysis (PCA) and then making a model out of the training set with SVM, including a automatic selection of parameters for the **RBF!** (**RBF!**) kernel used. The researchers concluded that this method had high efficiency for detecting intrusions and even outperformed other SVM based solutions. For DoS there was a 0.9940 detection rate and 0.0015 false alarm rate.

Variations of kernels are showing up regularly, and they have been tested with the KDD99 dataset for their accuracy in intrusion detection. For instance the Wavelet kernel has been tested and proved to be useful, with an accuracy rate of 93.1% and a false positive ratio of 8.7%, while the traditional **RBF!** kernel achieved only an accuracy of 86.1% and 20.1% false positives and Gauss kernel sat in the middle with 89.7% accuracy and 14.3% false positive ratio [19]. These results were made based on only eighteen of the forty one features of each observation in 10% of the KDD99 data.

2.2.2 Anomaly Detection in Network Traffic

Anomaly detection algorithm, or outliers detection have seen much interest in several fields, like network intrusion detection and fraud detection, mainly due to their focus on detecting, inside a big amount of data, small occurrences that are different from the rest. They include algorithms like clustering, some neural networks and density based techniques like KNN. The usual problems and set backs for this type of unsupervised learning is how to efficiently find outliers and how to interpret them. To help research, ELKI is a famous and easy to use open-source software that performs this types of detection and other unsupervised methods for data mining [20].

2.2.2.1 Outlier Detection

A self adaptive IDS framework was proposed in [21]. Using Snort, the system was able to, firstly, apply all the community rules for incoming traffic. Secondly, the packets were studied for outlier detection, and if the number of not normal packets found were over a certain threshold, they would go through another round of data mining, this time with clustering and association algorithms, in order to form a Snort rule that would be automatically joined with the other rules for detecting intrusion, thus being a self adapting framework. They had very good detection for DoS and probing

attacks, 100% in both, but the platform failed to find out about brute force and password guessing attacks and even root access, with a less than 50% detection rate in all test cases.

2.2.2.2 Clustering

Clustering analysis is another subject of study that can be used for anomaly-based detection. In this case, the entries are grouped into clusters, with each observation inside a cluster having high similarity between them, but maintaining a low similarity between different clusters. Graph-based clustering have been proved to achieved 95.3% detection rate with a false positive rate of 2.08% [22] with a subset of the KDD99 data. In this work is also mentioned that the major problem encountered was the complexity of the algorithm if the dataset was to be increased in size.

2.3 Intrusion Detection Systems

One principal part of this dissertation are IDS, which are in charge of protecting their users against the growing rate of attacks and defending them against external and internal threats. They play a major part in security and can be characterized in three different aspects: how they find intrusion, what action is done when said intrusion is found, and lastly where they are applied. Understanding these distinctions is essential to correctly implement any IDS solution. In this section is explained each of these different aspects, how they interconnect, and work related for this different types of detection is also presented.

As for how IDS work, they can either be Signature Based IDS or Statistical anomaly-based IDS. The first type has access to a database with known signature attacks and the software compares the incoming network packets with those in the database to determine if the traffic matches, or not, to a cyber attack and then can act accordingly. Snort is an example of a signature based IDS and is further explained in Section 2.3.1. Signature based IDS will struggle to protect the user against first day attacks, advanced persistent threats and even attacks that just change some bytes in the payload of the traffic to purposely pass through the filter as a legitimate connection. They also have problems with the amount of false positives that are triggered and their efficiency drops if there is too much data to process, called data overload. In the other hand, anomaly-based IDS make use of data mining algorithms to classify traffic as legitimate or an attack. This detection type has better results where signature-based ones fail, meaning that attacks that have no recorded signature can be detected by not falling under the patterns for what is considered normal activity in the network and data overload is easily handled by data reduction techniques.

When a IDS detects an intrusion, it can handle that situation in different manners. Usually there are two distinct approaches: a passive or a reactive IDS. Passive IDS limit themselves on simply alerting the network manager that a possible intrusion was detected, by writing this information on a log file or starting an alarm, the network manager will then have to look at this logs and take action accordingly. The reactive way actually blocks, or denies, the connection automatically. This type of IDS, also known as IPS, can also prevent even more attacks from

happening depending of its configuration, for example it can block any communications from a certain IP address that was considered as an attacker.

There is one more way to define IDS, and that is in which scenario they are employed, being either Network-based Intrusion Detection Service (NIDS) or Host-based Intrusion Detection System (HIDS). In an HIDS environment security software, such as antivirus, firewalls or Snort, is installed in every single computer that belongs to the network and this software work independently, aiming to detect and prevent attacks, either external or internal, that target any machine. NIDS are employed in well defined access points, such as gateways, that are key to maintaining the availability of the company using them. In this case, the main goal of using NIDS is to protect important services and control the attacks that come from the outside of the network.

Because cyber attacks are always evolving, IDS also need to adapt. Researchers are trying to enhance IDS in order to better protect users and diminish the negative results that an attack can do. One of the common problems of IDS are, that in a busy network, a signature-based implementation with regular rule sets can produce a enormous amount of alerts, some more important than others, and analyzing all of them is impossible for a network administrator. Usually the solution is to apply correlation and clustering to the alert data. In [23] a solution to this problem was arranged, by developing an extra software, called EDGe, that calculated the correlation between the alerts in order to detect recurring behavior, and using this information to find hosts infected with malware that could compromise the integrity of the network. Testing with a nine month long IDS alert logs proved that EDGe could find 60% more infections than other solutions, while keeping the false positives at 15%. Another approach to this problem is shown in [24] where the alerts were captured from a university backbone, with Snort, and then used to study what types of correlation would produce better and faster results, while implementing a distributed platform to perform this analysis, taking advantage of hash tables to store results making the process even faster.

Because networks are very distinct from each other, in size, servers, software or hardware used, no IDS implementation is ever going to be the same and have the exact same results. For this reason, different IDS frameworks have been developed, with several applications in mind, for instance there as been interest in using an IDS to protect mobile ad hoc networks [25]. Another framework was proposed in [26] where the goal was to developed a scalable IPS, where both HIDS and NIDS were applied, while also making use of a correlation software to analyze logs. If an attack was detected in any host, that information could be propagated to the rest of the network, in an attempt to block the attacking IP address.

2.3.1 SNORT

Snort is an open source signature based IDS that is capable of real time traffic analysis and packet logging [27]. It is a highly used intrusion detection system because of its very lightweight software, has simple implementation and a great community that updates the filter rules on a daily basis with very good efficiency in detecting and preventing attacks with documented signatures.

Although Snort was made for detecting intruders and attacks, it can also work as a packet logger only. To make the detection system work, one can simply write rules, or download the rule

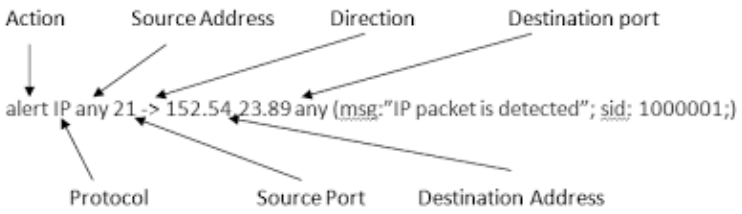


Figure 2.3: A standard for Snort rules [28].

set from SNORT website, and the software will compare all packets that pass through an interface with all the rules active at the time.If any of the packets are matched by a rule the software can either alert the user, log, drop, ignore or even reject the connection based on what was specified previously in the rule. Because of this ability to block certain communications Snort is also an IPS. The language for the rules used is simple and efficient and the structure for this rules can be seen in figure 2.3. In this case, if any IP packet addressed to the host 152.54.23.59 is found it will be alerted with the message "IP packet is detected". Everyone can make their own rules or use the ones supplied by Snort, completely free.

Snort is mainly composed of three systems: the packet decoder, the detection engine and the alert and logging system [29]. The packet decoder has to read the raw packets and decode them in the different OSI layers. Speed is a necessity in this system and its main function is to mark packets for later analysis. The detection engine comes next, and it is responsible for searching the packets for the signatures presented in the rules files. For efficient search, the rules are previously condensed into two lists: Chain Headers and Chain Options. After that each packet passed is compared recursively against the rules. The final system is responsible for producing alerts to wherever the user specified (to syslog or to an alert file for instance) and logging packets that triggered rules, if that is the case.

Because the community behind this tool is so strong and willing to make Snort a better IDS constant improvements, projects and research is always being done. These studies are usually targeted at a specific Snort problem. For instance, Snort can not detect attack that are not written in its rules. So, if a new attack is discovered, the community or researchers will, as soon as possible, find a pattern and a signature for the new attack and write a rule so that the users can be safe against it. Common old attacks are no exception either, and they see improvements in already existing rules or new, more effective ones are developed instead. In [30] Structured Query Language (SQL) Injection attacks were studied and five new Snort rules were written that could outperform the freely available ones but with a slight rise in false alarm rate.

To tackle the limitation of finding newly discovered attacks, or small variations of already detailed ones an integration with K-Means clustering algorithm was proposed in [31], therefore joining both anomaly and signature based detection. With this system, and using 17 cluster for the algorithm, they could detect two new attacks that were not written in rules with no false alarms.

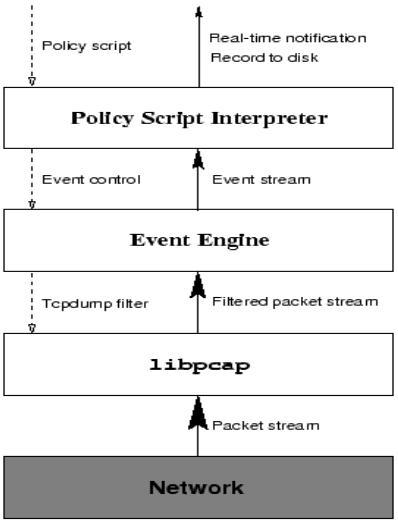


Figure 2.4: Bro NIDS layers [33].

2.3.2 Bro Network Security Monitor

Bro is an advanced open source security platform for intrusion detection exclusive to Unix machines [32]. This NIDS was programed according to important goals: high speed and large volume monitoring, no dropped packets, real time notification of what is happening in the network and the previous acknowledgment that the platform will be a target for attackers. The normal behavior of this network tool can be simply explained in layers, represented in figure 2.4.

Firstly it uses libpcap to capture the traffic on the network while applying some filters at the same time. After that the previously recorded packets make their way to the event engine, where Bro checks if all packets are well formed, if the calculated checksums match and IP fragments are also reassembled. If any packets are not correct, they will be immediately dropped and an event will be generated and passed to the next layer. This engine is also responsible for analyzing TCP and User Datagram Protocol (UDP) connections. Finally the policy script interpreter is in charge of outputting to the user real time notifications and create action policies based on the events passed by the event engine.

In default configuration, Bro sniffs the network for several protocols, such as Finger, Telnet, Ident and File Transfer Protocol (FTP). It also has behind a very supportive community that often release scripts to detect dangerous exploits or vulnerabilities, such was the case with Heartbleed and Shellshock.

2.4 Distributed Computing

Technology is always evolving. Computers are no exception and the amount of memory and processing speed are growing each year. But even then, some tasks can not possibly be done, in due time, by a single computer. Distributed computing is the answer to this problem and it can coordinate several computers, called a cluster, in order to perform a job simultaneously, therefore

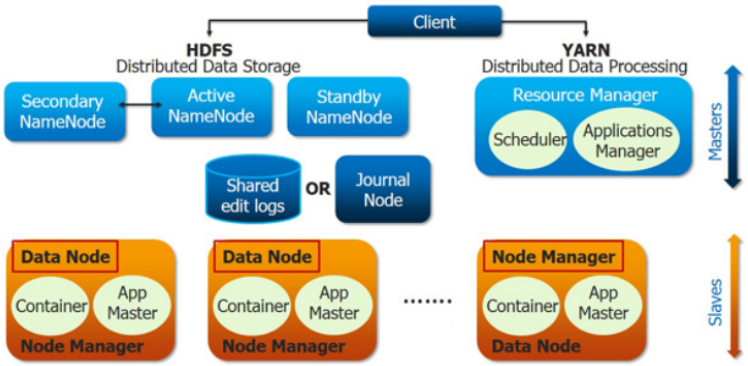


Figure 2.5: The Hadoop framework [34].

finishing the task quicker. It is necessary to have a middleman managing the interactions between the computers in the cluster. Hadoop is one of the most used software for distributed computing, as such, it helps the success of companies like Ebay, Facebook and Google [5].

In this section important research done recently in the field of intrusion detection while taking advantage of distributed computing is presented. In Section 2.4.1 is presented a brief Hadoop introduction, along with some of its uses and recent work in security and intrusion detection fields.

2.4.1 Hadoop Platform

The Hadoop framework is mainly composed of four modules: the Hadoop Distributed File System (HDFS) in charge of distributing the data efficiently through the cluster; MapReduce, that handles the parallel processing of any queries done by the user; Yet Another Resource Negotiator (YARN), responsible for managing the clusters; and the Common module, with all the utilities to support the previous modules. An overview of the framework provided by Hadoop can be seen in Figure 2.5

Hadoop, being a open-source project, has had much work to improve its performance and make it an even better distributed computing platform. MapReduce, one of Hadoop's key modules that was developed by Google, has a master-slave architecture and is divided in three steps. The first one is the map step, where the working nodes, in parallel, apply filtering and sorting to the data according to the query done by the master node, followed by the shuffle step that redistributes the data that was mapped before, and finally the reduce step, which shows the final results of processing a certain query. Because of the importance of this module for the global efficiency of Hadoop, MapReduce has seen continuous effort to better map and reduce the data. In [35] an alternative shuffle algorithm was developed, that achieved an average of 8.94% speed increase, with the best result being 12.41%, testing with a word counter job. A correlation based block placement algorithm was proposed in [36] that aimed to address a not optimal MapReduce performance when doing multiple file queries, obtaining a faster response time and processing speed when compared with the default policy. A cloud bursting solution to process intensive MapReduce jobs on Internet

traffic was studied in [37] where the authors achieved 50% less amount of traffic sent to the cloud bursting than when using the normal scheduler.

2.4.2 Hadoop for Intrusion Detection

Hadoop has been proved to be capable of analyzing IDS logs faster than a single node, when dealing with just four gigabytes of connections, directly captured with SNORT, and a five computer cluster, the processing speed was up to 398% [6]. This rising in processing speed will be even more important when dealing with Big Data size files, allowing for a more deep inspection of all traffic recorded. This was one of the motives to apply this extra knowledge that can be gain with Hadoop and distributed computing to specific attacks, like detecting phishing webpages [38].

In [39] Hadoop was used to quickly retrieve and analyze the large data sets that originate from alerts provided by honeypots (a system or server that acts as a trap to intruders). The researchers created a system for dynamically deploying honeypots that they called Honeydoop which had four modules: Intrusion Detector (with Snort), Enumerator (finds details of the probed system), Auditor (running important scripts) and Virtual Honeypot Manager (manages all honeypots). With testing was proved that they could generate and process honeypot alerts better than the traditional way of deploying a honeypot.

There have been experiments with the KDD99 dataset and Hadoop, and for a clustering based algorithm, above 90% detection rate was achieved with 25 k-means cluster, using twenty thousand entries [40]. Unfortunately with more than one hundred and fifty thousand connections it would produce inferior results, meaning there is still much work to be done to increase efficiency in finding intrusions with large data sets.

2.4.3 Packetpig

Packetpig [41], a network security monitoring platform, is a tool that uses Hadoop in order to store large amounts of full packet captures and for further processing, while also using Apache Pig for ease of programming and extending the project to the most people possible. It completely relies on open source tools and even has Snort integrated.

This tool can detect intrusions based on signatures, look for packets that have "never been seen" in the network and it can learn when the attacks happened, what were the targets and the attacker, and produce triage incidents. Of course, this is not done in real time. The packet captures are stored in a computer cluster and then MapReduce jobs are started by the user using Apache Pig easy language.

Packetpig's core features include accessing any IP header field in the packets, protocol analysis, study network flows, threat analysis with Snort, it can track IP addresses to country and even to latitude and longitude and fingerprint operating systems. This is all possible thanks to the java written Pig Loaders [42].

2.5 Discussion

In this chapter the state of the art solutions for capturing network traffic, different IDS and possible enhancements to them, Hadoop and related work were all presented. It is possible to conclude that Big Data is trending, because of the ease of setup and the large amount of information one can collect, and that there is much work being done to improve the effectiveness of data mining techniques when searching for intrusion detection.

IDS were also subjected to research and, upon all results gathered, Snort was concluded to be a very strong tool, that sees much use trough companies and normal users worlwide. It also has some imminent problems as far as signature base IDS go, and it can capture traffic and analyse it in real time. It also works with full, already saved, packet captures losing the real time detection, and in this form it can consume much time to process a pcap, especially if a large amount of rules are loaded or if the size of the capture is substantial.

In this dissertation thesis, Snort was joined together with Hadoop and its performance studied and evaluated. There were found papers and works about using Hadoop to process Snort's alert files but projects that distribute the processing itself and study the performance obtained were not found. The closest thing is packetpig that has a special loader that uses Apache Pig to search for intrusions with Snort in full packet captures located in a Hadoop cluster. Packetpig, while being completely scalable, needs the pcaps already captured and located in the cluster. Unfortunately there exists a lack of studies for this tool specially dealing with the Snort processing. Because of this there was a need to find out exactly how can Snort be parallelized while mantaining its detection capabilities.

For the previously mentioned reasons, a completely automated tool was developed. This tool, called The Hunter and detailed in Chapter 3, is scalable to any cluster, has Hadoop integration, captures the traffic and distributes the Snort processing on its own, while also fetching and reducing the alerts produced.

Chapter 3

Developed Platform: The Hunter

The major objective of this work was to distribute the processing of Snort through several computers instead of using just one machine that does all the job (sniffing, saving and analyzing traffic) and study the impacts of this distribution, both on time consumed and efficiency obtained. With that in mind, the idea of making a platform that could perform this task in a completely automated and scalable way arose, and The Hunter was born. In the end, with all scripts developed working, The Hunter was configured to have a cycle behavior. In this cycle it would capture real time traffic, and at the same time search for intrusions on the packets that were captured in the cycle before. The time for each detection cycle is completely up to the user.

A special use case was also implemented in the developed platform in which the detection cycle would run in parallel with a previously configured MapReduce job. If this task could be performed without hurting the detection time and efficiency of Snort it would mean that the deployed computer cluster could also be used for other user chosen MapReduce jobs, and not be made specially for Snort, therefore giving the option to anyone using this proposed platform for also using the cluster as they see fit for their own benefit, while also using Snort to its full extent, finding out if their services are being targeted by any attacker.

The Hunter is made of several different scripts, written in bash, Ansible, Java and Python. Ansible was the chosen tool to help distribute commands and scripts in a parallel way to the computers involved in every operation [43]. There are two special scripts that join everything together and are key for the correct functioning of the platform. These are The Packet Hunter, in charge of distributing Snort's processing and the detection cycle, and The Capturer, that captures traffic in special ways, and they were both developed in Python.

The Hunter folder is composed by The Packet Hunter script and other directories, such as a playbook directory where all Ansible scripts and Snort configuration files and rules are located. There is a special directory just for The Capturer script and another for the classification program used with MapReduce and all its dependencies. Finally there are also two directories where the Snort alerts and MapReduce results are stored, for further user inspection.

In this chapter the initial setup necessary to run The Hunter will firstly be presented. There are also some thoughts given to nuances and problems that originated with The Capturer on sec-

tion 3.2. Up next, in section 3.3, is a detailed explanation of The Packet Hunter with use cases and the algorithm for this python script. Some attention to smaller, but nonetheless important scripts is given in section 3.4. Finally a quick overview of this chapter and closing remarks are presented in section 3.5.

3.1 Initial Setup

Before presenting the most important scripts that are part of this hunter tool, it is mandatory to give information about initial required setup. Because a distribution of Snort processing is going to be made, a computer cluster is necessary. Openstack was used to deploy clusters easily, with already configured Secure Shell (SSH). The deployed clusters are composed by a Master and a certain number of Workers, and all of them are actually deployed virtual machines. One problem with this setup is that the cluster can be easily taken down with all data that was stored being lost, for instance if there is a power cut. After that Snort needed to be installed and configured on every Worker of the cluster.

Ansible was a big part of this project as it helped distribute commands and scripts simultaneously to the cluster, sometimes it was even used just to run commands and scripts on the Master. This means that Ansible has to be installed and properly configured on the host where The Hunter folder is downloaded to. This Hunter folder has inside it all scripts that make this autonomous and scalable platform possible. From this point on, the computer that has this folder will be called the Starting Host in order to prevent confusion with the workers and the master in the computer Cluster. Installing Ansible is simple, one can do it from any package manager, and for the configuration it is only necessary to edit the file host inside Ansible directory and include the group "*workers*", containing the IP address of all the workers in the cluster, and the group "*master*", with the proper IP address. For security purposes Ansible should be configured with SSH keys, not passwords, and The Hunter only works with that setup. Again, if using OpenStack it is possible to configure all computers in the cluster with SSH and download the corresponding key file to the Starting Host. Obviously the Starting Host has to have a connection to the created cluster and a 1000Mb/s connection is recommended. If using less, the transfer of files between the Starting Host and the Workers will take much more time and will become a bottleneck for the platform.

The Packet Hunter and The Capturer scripts are both written in python, therefore it is also necessary for python to be installed on the Starting Host. The used python version was 2.7.10. This Starting Host also has to have a Linux distribution, due to the configured commands on most scripts developed being exclusive to Linux.

One of the use cases for testing the developed tool involved starting a MapReduce task at the same time of Snort's processing to access to workload on the computers. For this purpose Hadoop and its core modules, like HDFS and MapReduce need to be properly configured on the cluster. If using OpenStack this is automatically done.

3.2 The Capturer

The Capturer is the python script in charge of capturing, storing and sending the traffic to the Starting Host, where The Packet Hunter is working. This script is always capturing traffic in cycles, with each cycle having a certain time interval that is chosen by the user. It is important to note that this script is not essential for the execution of The Packet Hunter, but it helps in the way of periodically bringing pcaps to be analyzed by the main script (explained further in section 3.3). Even though using The Capturer is not mandatory, because any user of this developed tool can capture traffic as he/she may seem fit, it was used in every test made in this dissertation thesis so it is only right to give some information about how it works. It also has some nuances and challenges that are going to be discussed further in this section.

The first important step is to explain what arguments this script is expecting and how they change its performance. Firstly it is necessary to tell the program if it is going to run in a local configuration or a remote one, and this step happens so The Capturer knows where to send the packet capture, when they are ready, and it will always send to the working directory of The Packet Hunter on the Starting Host. Usually, for testing purposes, both scripts (The Packet Hunter and The Capturer) are working in the same computer, therefore the local configuration is used. But if one needs to capture in one specific machine in the network and The Packet Hunter is working on a different computer, then the remote configuration is necessary (and previously setup SSH connection). In this program one can do this by passing the argument "remote" or "local" followed by the location of where the captured pcap is to be sent. The only thing left to tell the script is, obviously, for how much time it will run each capture cycle. This is done in minutes.

Just to exemplify for a local configuration with a 30 minute capture cycle one can start The Capturer with the command "*sudo python TheCapturer.py local /home/mike/Documents/ 30*", where */home/mike/Documents/* is the location where The Packet Hunter is working. In the previous example, both scripts are working on the Starting Host. On the other hand, if one uses the command "*sudo python TheCapturer.py remote ubuntu@192.168.109.228:/home/mike/Documents/ 30*" this will put the script in remote configuration and use the IP address and user to copy the pcap to the remote directory specified, that is where the Starting Host is working.

Upon executing the script, and after the arguments have been correctly parsed, the traffic on a specific interface is now being captured with TCPDump for indefinite cycles, with each cycle taking a certain amount of user specified time and ending when a terminal signal is received (in the form of ctr+c). Immediately after ending a cycle, another one begins so that no packet is lost. There is also a thread that starts in charge of sending the previously recorded pcap to its final destination, whether that may be with just a move command (local configuration) or with a secure copy (remote configuration). A success file is also sent to the working directory of The Packet Hunter telling it that a processing cycle can now take place. If the user terminates The Capturer in order to stop it from recording the internet packets, the program will finish the capturing cycle it is on, and after sending the pcap and success file, it will also send a end of capture file so that The Packet Hunter knows that no pcaps are to be expected after the one just sent.

One of the big problems faced here was that the file that results from a capturing cycle is, eventually, going to be sent over the network while there is another capturing cycle going on. Whether the previously captured pcap file is now sent to another PC with secure copy (with remote configuration), or later when the script The Packet Hunter decides to distribute the file (when running local configuration) this transfer is going to be captured by TCPDump because it is always running in cycles. Therefore the problem of, with each cycle, getting pcaps with bigger and bigger size arose. The solution found to this problem was to apply a filter to TCPDump in order to prevent the capture of any packets between the Starting Host machine and the cluster computers (which are deemed safe) or between the Starting Host and the machine where The Packet Hunter is running. For instance if the command "*tcpdump -i eth0 not host 192.168.109.10*" is used, all communication between the machine running TCPDump and the host with IP address 192.168.109.10 will not be captured.

In figure 3.2 is presented the code responsible for executing the TCPDump command, when dealing with local configuration. In this case it is important to ignore the packets exchanged with all the cluster computers. It is also good to note that, when starting TCPDump with sudo privileges the linux operating system actually starts two processes. In order to stop the capture once the user defined time has expired the spawned processes are grouped together and killed together.

```
# command to run as a subproccess
cmd= "sudo tcpdump -i " + interface + " -w "+ fullPcapName +' not host ' + master

while (i< len(workers)):

    cmd = cmd + ' and not host ' + workers[i]
    i+=1

pro = subprocess.Popen(cmd, stdout=subprocess.PIPE, shell=True, preexec_fn=os.setsid)

#leave running for timeInSeconds time
sleep(timeInSeconds)

#kill all processes that spawn so capture ends
os.killpg(pro.pid, signal.SIGTERM)
```

Figure 3.1: Making the TCPDump command on The Capturer

3.3 The Packet Hunter

The Packet Hunter is the most important script developed in this thesis. It joins together other scripts for the common goal of distributing the Snort processing through all the computer workers in the cluster, and even gathers the results obtained in the form of alerts to the Starting Host.

The algorithm for The Packet Hunter can be seen in figure 3.3. In a simple way, when the Starting Host executes this script, there are some configurations and setup that take place on the workers and the master of the cluster. After that, The Packet Hunter waits for a pcap to be processed and then enters in the detection cycle. In this cycle it will divide the pcap received in equally sized parts according to the number of workers the cluster has, send each part of the original pcap to each worker and then run Snort from the workers, in a parallel way, to analyze the previously

copied packet captures. The only thing left in the detection cycle is to analyze the alerts produced by Snort and reset some changes that were done during this cycle.

The problem that is approached is that, when running Snort with a significant amount of rules from a single host the processing time of a certain capture is significantly slowed down. It is even worse if Snort is working in real time, and it gets unbearable if there is also a capture of traffic taking place, many packets are dropped and not processed. Even Snort acknowledges this and that is why they comment out most rules they give to their users by default, defending a "Connectivity over Security" policy when using Snort in real time, for most users [44].

When using this tool, however, the Snort processing is distributed to several computers with the policy "Security over Connectivity", supposedly allowing an overall faster processing whilst also maintaining a good efficiency, which is what we are trying to prove. The downside is that the real time intrusion detection is lost, and instead a delayed one is done.

In section 3.3.1 is explained the most important functions that are taking place when using The Packet Hunter with the sole purpose of distributing Snort workload. A special use case is also detailed in section 3.3.2, in which the pcap that The Packet Hunter receives is also sent to HDFS and a MapReduce function is executed at the same time of Snort processing.

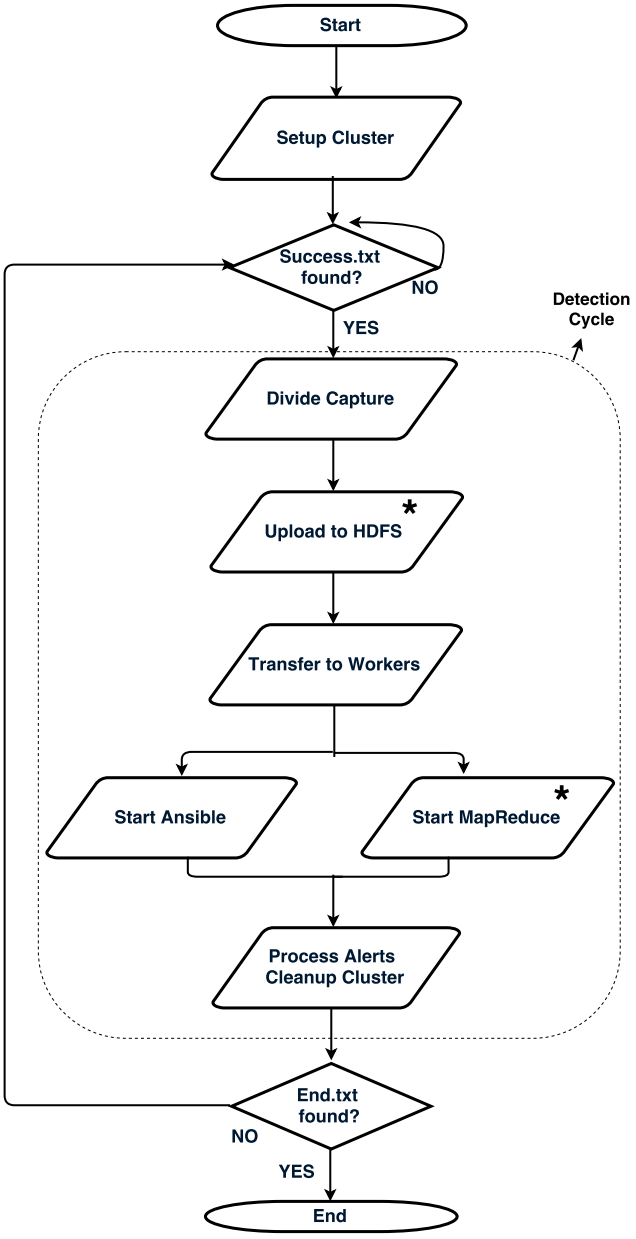


Figure 3.2: The Packet Hunter Algorithm

3.3.1 Normal Use

This python script has three distinct steps. The initial set up for the master and the workers in the cluster, the waiting for a packet capture to process and the detection cycle.

The initial setup starts as soon as the script is executed. It needs sudo privileges and one can use the command "*sudo python ThePacketHunter.py 0*" on a Linux terminal to start the program. The only input the user needs to make is the first argument passed and it can be a "0", "1" or "2". With a "0" the program will only process the pcap with Snort. On the other hand, with a "1" it will only process the pcap with a MapReduce job. The final option "2" will do both tasks. Upon doing this, the script will attempt to reach the IP address of the workers and copy to each one a Java

program that is in charge of running a terminal command in order to start Snort IDS and return some statistics on processing time. A connection to the master of the cluster is also attempted and, if successful, the jar file that contains the MapReduce Classification function, based on Parallel Packet Processor [45], is transferred. After this the connection to all computers in the cluster is confirmed, and everything is set up for the next scripts steps.

At this point the detection cycle has almost started. The first thing that marks this cycle is the moment when a success file is found in the working directory. This file, that can be sent over by The Capturer script, has in it written the number of seconds the capture of internet traffic took, and it symbolizes that the pcap to be processed is now ready and located in the Starting Host, where The Packet Hunter was executed. This means that the detection cycle can start and it will now take the full capture and divide it equally in size for as many workers there are in the cluster. This is done thanks to tcpdump -C flag that can write a capture while keeping it under a user specified amount of bytes. The corresponding python code in charge of calculating the amount of bytes for the -C flag and the execution of the TCPDump command can be seen in figure 3.4.

```
def dividingCapture (howMany, name):          # fairly divide original pcap into equal pieces for all workers

    ownership = os.stat(name).st_uid

    if (ownership != 0):

        changeOwner = 'sudo chown root ' + name
        os.system(changeOwner)

    sizeInBytes= os.stat(name).st_size
    newName= 'broken'+name
    onePieceSize= ((int(sizeInBytes)/1000000)/howMany)+1

    # -C flag takes million bytes as input
    cmd= "tcpdump -r " + name + " -w "+ newName + ' -C ' + str (onePieceSize)
    pro = subprocess.call(cmd, stdout=subprocess.PIPE, shell=True, preexec_fn=os.setsid)
```

Figure 3.3: The Packet Hunter function for dividing the pcap

With the pcap divided it is time to send each piece to each worker. This is done with secure copy, without any threading. This can mean that, with a 1000Mb/s connection, the transfer rate of each individual piece of pcap to a computer in the same LAN can achieve values as high as 70MB/s. If threading was implemented, even though the transfer of all pcap pieces could be faster, the high transfer rate will diminish according to how many threads are used and if many threads are started at the same time the computer running the script can be slowed down.

The next step of the detection cycle is to start Snort on each worker to process the previously copied pcap piece. This is where Ansible comes in, as it helps to start different operations, with a certain order, in a distinct set of computers. A script with several Ansible commands is called an Ansible playbook. The Ansible playbook used in this stage is configured to run on the workers only, and has the task of running Snort, outputting the Snort processing time of each machine and fetching the produced alerts to the Starting Host where The Packet Hunter is working. This and other Ansible playbooks will be detailed further in section 3.4.

Finally, the cycle is almost over and the most intensive processing is done. What happens next is a small convenience for the user in which the previously fetched alerts are reduced and their

content is summarized on screen. This alerts are also saved on a special folder (named "*oldAlerts*") with a timestamp in order to facilitate future access for the user. An Ansible playbook is also used for cleanup purposes in the end of each cycle, for instance deleting the parts of the pcap of each worker and resetting the alert file so that no alerts are repeated. Now, the program will either wait for another success file in order to repeat this cycle, but on a new pcap, or end all processing if a end of capture file is found in the working directory, which can be sent over by The Hunt Begins program. Some time statistics are also printed at the end of each detection cycle.

3.3.2 Use with MapReduce

In this use case The Packet Hunter program was modified in order to work with Hadoop core modules: HDFS and MapReduce. This was done with the intent of studying if running MapReduce jobs in the cluster will affect in any way the Snort processing of packets, when running simultaneously, and how much more time was spent doing these new operations.

A packet classification program, based on Parallel Packet Processor, and developed by MIEIC students at FEUP, was used to run MapReduce jobs on the packets collected. This job classifies network traffic according to their protocol, but its efficiency in that was not studied. The real objective that was tackled by adding this step was to study what happens to the cluster when both MapReduce jobs and Snort are running at the same time.

As for the python program itself, there were two additions made, that would only happen if the user, when starting the script, would pass the argument “1” or “2”. The first addition was implemented as soon as the detection cycle began: the WebHDFS REST Application Programming Interface (API) was used with the objective of uploading the complete pcap to HDFS. This makes the stored packet capture available for future reference and also allows the execution of MapReduce jobs on the uploaded file. It is important to note that this operation will, probably, be a bottleneck for the script and consume some detection cycle time, when compared to other steps of this cycle, but this was a necessary step for executing MapReduce jobs and, therefore, studying the proposed program. This upload happens right before the secure transfer of the capture pieces to each worker. No threading is involved in this step.

```
def copyToHDFS(pcapName, IPhdfs, cycle):            # copy full pcap to HDFS

    cmd = ('curl -i -# -X PUT -L "http://' + IPhdfs + ':50070/webhdfs/v1/' + pcapName + str(cycle) +
        '?op=CREATE&user.name=hadoop" -T ' + pcapName + ' | grep Created')
    os.system (cmd)

def startMapReduce(cycle, pcapName, sshKeyLocation):

    line7 = '    command: sudo -u hadoop \/opt\/hadoop\/bin\/hadoop jar p3lite.jar p3.runner.TrafficAnalyzer -r\/'
    sed = "sed -i '7s/.*/" + line7 + pcapName + str(cycle) + "/' playbooks/runClassification.yml"
    os.system(sed)

    ansCmd ='ansible-playbook playbooks/runClassification.yml --private-key ' + sshKeyLocation + ' | grep failed'
    os.system(ansCmd)
```

Figure 3.4: The Packet Hunter functions for uploading to HDFS and starting MapReduce

There was also added a new Ansible playbook for starting the MapReduce job in the master. Even though this is only working in one host, it is simple and effective which justifies the use.

This playbook starts at the same time as the Snort processing and it essentially does two things. It sends the Hadoop command in order to start the MapReduce jobs on the previously uploaded pcap and then, when the jobs are over, it fetches the results from the hadoop logs located on HDFS all the way to the computer executing The Packet Hunter script, saving them on the folder named "*oldParts*".

The python functions responsible for the upload to HDFS and the start of the Ansible playbook that starts the MapReduce operations can be seen in figure 3.5. The REST API for HDFS can be accessed through the port 50070 on the IP address of the master computer. A curl command was put together to connect to this port and IP with the intend of uploading the file, using the operation create of the API and, in this case, the user 'ubuntu', coded in function *copyToHDFS* . Because the program is running in cycles, the Ansible playbook that executes the hadoop command on the master for starting the MapReduce jobs has to be changed each cycle with the new pcap name. This and the terminal command for starting the playbook are part of the function *startMapReduce*.

3.4 Other Scripts

There are several other scripts, written in Java, Ansible and even bash, that are needed for the correct behavior of this tool. Some of them run only once, on cluster setup for instance, and others run every single detection cycle. The most important ones will be highlighted next and briefly explained.

A set of scripts were made in bash to deploy and configure Snort in the workers of the cluster started by OpenStack. These scripts would install all dependencies for Snort, like package bison and flex, and download and install the latest versions of libpcap from tcpdump webpage, and also daq and Snort from Snort's main page. All these three downloads are a must if Snort is to properly work. After the successful installation of Snort, it was also needed to configure some options and create some directories. Automating this task was a necessity because of how often it was necessary to restart the cluster. To run both configuration scripts in a newly created cluster, a Ansible playbook was also put together.

Because java is installed in all computers, they supported Hadoop after all, it was developed a simple java program, with just a few classes, in order to run Snort, gather the normal output from the console terminal, and from there extract the processing time by searching for the sentence "*Run time for packet processing was* " and print this information. Again, Ansible was used to distribute the execution of this java program through the cluster and also fetch any resulting alerts that could be produced by Snort.

There are some Ansible playbooks that were used during this dissertation thesis that are included in The Hunter even though they are not used in any way during the execution of The Packet Hunter and The Capturer. One of this scripts changes the Snort rules and configuration file on all Workers, which can be useful for the user if there is a need to add or remove certain rules.

There is a need to edit configurations on specific scripts, like the IP address of the workers, or the directory where the SSH keys are located, but this is all detailed on read me files in each folder so any user can quickly adapt the tool to his working environment.

3.5 Summary

In this chapter the platform developed was introduced. The architecture of The Hunter folder was presented and the algorithm for The Packet Hunter, the most important script in the tool, was explained. This was all done with the intent of tackling the problems that results from running Snort in a single machine. This problems can vary between slow performance of the computer used and the lack of intrusion detection by Snort. With the distribution of the processing, using Ansible, we are able to load as many rules as possible into Snort so it has better chances in finding most different types of attacks. This has a negative effect, because it can lead to more false positives in the alerts and the detection done is now a delayed one, therefore it is up to the user to understand what went wrong after the attacks have already happened, or are still happening.

Chapter 4

Cyber Attacks and How Easy They Are

Snort, one of the best and most used intrusion detection system, is the main focus of this dissertation thesis. As such, this strong capacity for detecting attacks needs to be tested in our distributed Snort solution, presented in chapter 3, in order to prove its worth.

Snort makes use of signatures for detection, and these are written in Snort's many rules files. It was important to test, to a degree, Snort's efficiency in detecting intrusions when applying this rules to individual small pieces of the same pcap, because that is just what The Packet Hunter does. A comparison between this scenario and when processing the entire pcap in one machine is made in chapter 5.

There are many intrusions that can be practiced, and many different cyber attacks that can be made. In this work, due to the available time, it was not possible to perform an extended and exhaustive penetration testing. Instead just two types of attack were taken into account. The chosen types had to be a) significant and important nowadays, and b) easy to understand and replicate. This important requisites helped narrow down the search to SQL injection and DoS attacks.

SQL injection attacks and how to exploit this vulnerability will be explained in section 4.1 and Denial of Service attacks are detailed and demonstrated in section 4.2.

4.1 SQL Injection

According to OWASP top 10 the single most dangerous and common attack is command injection [46], for example SQL injection. These type of attacks happens when the user input is not properly escaped, sending untrusted data to an interpreter, and if this vulnerability is encountered it is relatively easy to exploit, due to a large amount of examples available online and automated tools like sqlmap [47], and when successfully exploited the impact is extremely severe: most things from the database can be dumped by the attacker.

The idea behind an SQL injection is simple: one tries to escape out of the original programed query and somehow replace or change it with a new one. For instance if there is a website that shows information about a movie the user searches, the query is going to look something like this:

"*SELECT * From Movies Where MovieName = userInput*" . If one searches for the movie The Deer Hunter the query should now be "*SELECT * From Movies Where MovieName = The Deer Hunter*" and the server would return all information it has available about this amazing movie. An attacker could also type in "*The Deer Hunter OR 1=1*", therefore creating the equally valid query "*SELECT * From Movies Where MovieName = The Deer Hunter OR 1=1*" which is always true and returns all rows from database Movies.

There are many types of SQL injection, but the concept in all of them is always the same. In this dissertation thesis was used a web server provided by itsecgames called bWAPP, or an extremely buggy web app, and this is a PHP application with a MySQL database made specially for free usage to anyone trying to improve their own pentester skills [48]. There is also available a custom Linux virtual machine, called bee-box, that already has bWAPP installed and configured. For practice purposes and packet capturing bee-box was virtualized with VirtualBox, therefore providing more than a hundred vulnerabilities to be explored, including many different variations of SQL injection.

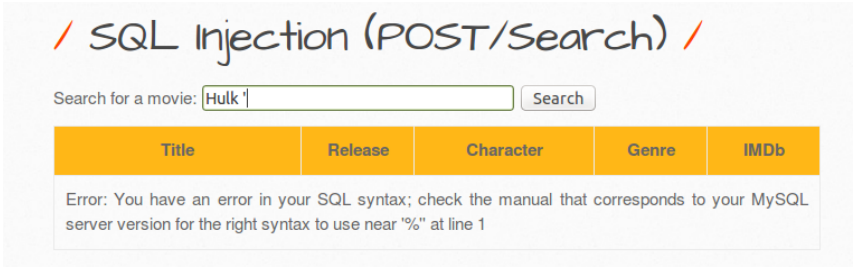
It was important to chose conceptually different SQL injection attacks to perform in order to test Snort in different ways, which lead to the exploit of six bWAPP levels, that will be detailed next.

4.1.1 Hacking GET/Search and POST/Search

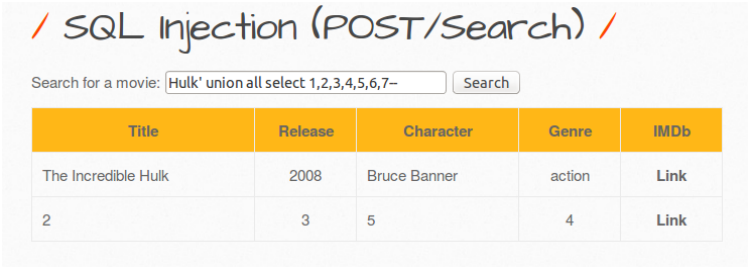
Firstly search boxes were attacked, with both Hypertext Transfer Protocol (HTTP) GET and POST requests. On bWAPP both attacks can be found on the page *sqli_1.php* and *sqli_6.php* respectively. One important difference between GET and POST is that the SQL query goes in the Uniform Resource Locator (URL) with GET request, but with POST requests the query is part of the body of the HTTP message. Even though the payload of the attack is located in different places of the HTTP message, the attack vector is equal for both situations.

Exploiting this search boxes was not a difficult task. If one tries to search anything followed by a single quote, the server will answer back with a SQL syntax error, demonstrated in figure 4.1a. A very common route was then taken, in which the union operator was used. This operator is responsible for combining the results of several select statements, working only if each one used has the same number of columns. With that knowledge the payload "*Hulk' union all select 1–* " results in a syntax error with the meaning that the number of columns for the two select queries are different. It is only a matter of incrementing this number to find out that the query "*Hulk' union all select 1,2,3,4,5,6,7–* " produces no syntax errors, shown in figure 4.1b.

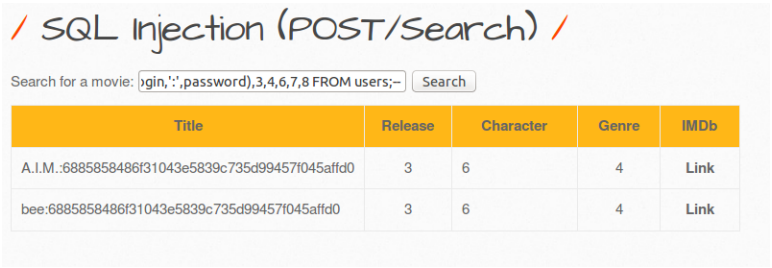
From this point on, it is just a matter of choosing what to ask the database. For instance, the payload "*-1' union all select 1,version(),3,4,database(),6,7–* " will output the MySQL version installed and the name of the database. It is also possible to get the names of all columns in all tables by accessing the metadata table information_schema. Using the attack vector "*-1' union select all 1,column_name,3,table_name,6,7,8 FROM information_schema.columns;–* " will produce this result. The previous attack allows us to know the name of the table where the user information is stored and the columns that make that table. Therefore the trivial dump of the login and password



(a) Quote syntax error.



(b) Finding the number of columns for Union query.



(c) Getting passwords and logins!

Figure 4.1: Learning SQL injection behaviour and exploits

of all users with "*-1' union select all 1,concat(login,':',password),3,4,6,7,8 FROM users;–* " is possible, which can be seen if figure 4.1c.

4.1.2 Hacking GET/Select and POST/Select

Select boxes were also victims of exploits, and just like with search, both GET and POST request were attacked. The attacks can be visited in bWAPP page *sqli_2.php* for GET and *sqli_13.php* for POST.

This time around the payload is still the same for both levels but the way of sending the attack is now very different. Because one can only select a movie between the ten choices available and then press the Go button to submit the request, it is not possible to insert here any attack vector. Luckily, when dealing with GET requests, the payload can simply be injected on the URL itself. With POST this can not be done and a way to change the body of this HTTP message, after submitting it, was needed. This was accomplished with the Firefox add-on Tamper Data, that acts like a proxy between the computer and the server, and allows the user to change any data that was already submitted.

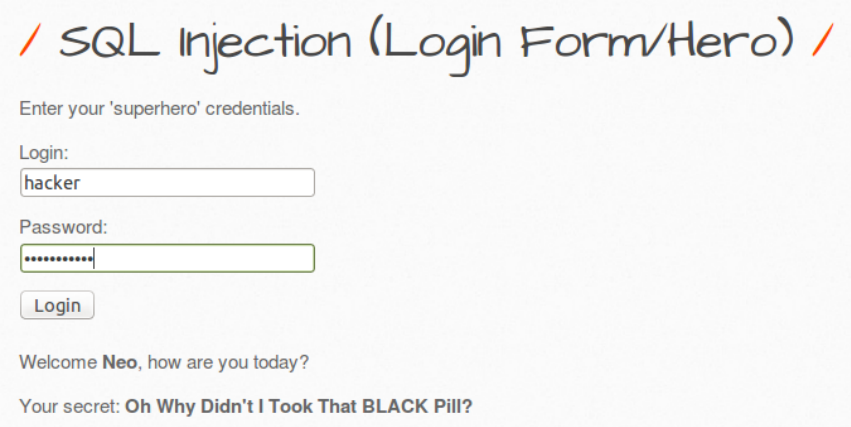


Figure 4.2: Bypassing login form.

The exploits used were almost exactly equal to the ones described on section 4.1.1, with the only difference being that there was no initial quote needed to escape the query. For instance, to dump the login and passwords the payload "*-1 union select all 1, concat(login,':',password),3,4,6,7,8 FROM users;–* " can be used.

4.1.3 Bypassing Login Form

The act of requiring a login and password before accessing some websites has always been used to prevent unauthorized access to private files and to protect users privacy. They are the very first security measure implemented in websites and services that need some kind of privacy. Because of this, the whole act of logging in has been victim to much penetration testing, and the consequences in bypassing authentication can be disastrous.

Testing the login forms for SQL injection is just one of the many vulnerabilities that can be found in authentication. In bWAPP a weakly implemented login form can be found by accessing page *sqli_3.php* and the objective of this level is to use SQL injection to change the query that validates the user and password inputted, in order to get authenticated by the server.

Exploiting this vulnerability was relatively easy, just a matter of understanding what the login query looks like in the server, and then send malicious input to alter its expected behavior. In this case, if a user sends his login and password the query is going to be something like "*Select * FROM users WHERE login='mike' AND password='12345'* ". If a bad intentioned user escapes out of the query with a quote he can bypass this authentication by writing in the password box the attack vector " *' or 1=1–* ". This will ultimately transform the query to "*Select * FROM users WHERE login='hacker' AND password=' ' or 1=1–* ". The attack vector changed the query to one that will always be true and results in the bypass of authentication. This exact inputs on the login form are demonstrated in figure 4.2 and it is possible to see that the hacker gets authenticated as the user Neo.

4.1.4 Experiments with Boolean Blind SQL injection

Blind SQL injection is a special type of vulnerability, much more time consuming to exploit than other code injections, that usually is encountered in web pages of servers that are not configured to output the data that was queried but instead show already configured messages. The injection itself is very similar to the cases seen in previous sections, the real problem is obtaining the results of the attacks. One of the ways this can be done is with true or false questions. If the database outputs absolutely nothing about the query one can try time based blind injection, which translates in telling the database to wait a certain period of time if the attack vector is true, or the opposite if it is false. If there are some different generic messages it is possible to use boolean based blind injection to study if the attack vector is true or not according to the message shown.

Only boolean based SQL injection was exploited, and the bWAPP page for this vulnerability can be found when accessing *sqli_4.php*. This web page is made of a search box that tells the user if the searched movie is available on the database. If the movie Goodfellas is searched the page will return "*The movie does not exist in our database!*". In the other hand, if Iron Man is queried the page now tells that this movie can be found with the response "*The movie exists in our database!*". This difference in responses is the key to exploit this vulnerability.

The escaping out of the original query is the same as that of previous sections, a simple quote does the trick. To confirm SQL injection one can type "*iron man ' and 1=1–* " and expect a true response from the database, which happens. The only thing left is to confirm a negative answer: the query "*iron man ' and 1=2–* " produces a false response.

From this point on it is just a matter of choosing what questions to ask the database. Usually this type of exploit is done with the help of automated tools. In this case only a few queries were made so there was no need to use sqlmap or to develop a script for this purpose. It was decided to find out what was the database name. With this in mind, first the number of characters in the name had to be found. If the query "*iron man ' and length(database())=1–* " is performed, the web page answers false. A simple incrementation of the number of characters leads to a positive answer when the size is five. Next the query "*iron man' and substring(database(),1,1)='a'–* " can be made to test if the first character is an "*a*". Again, when iterating this value the database reveals that the first letter of the database is a "*b*". By doing this process with all five characters the name of the database is revealed to be "*bWAPP*". This is a time consuming attack, and one should write a script for more advanced queries.

4.2 Denial of Service

Besides SQL injection, Denial of Service attacks were also used to test our solution. Most DoS attacks leave behind a large amount of evidence in the form of a spike in traffic, which is very useful for testing The Hunter because it increases the size of the pcap captured while also allowing a stress test on Hadoop and Snort.

A denial of service occurs when a certain service, server or machine is made temporarily unavailable to its users. This can occur in very different ways and in distinct layers of the OSI model. Usually there are specific ways to protect against these attacks, but because there are so many types of DoS it is also close to impossible for a user to be completely safe. Even big corporations like Microsoft and Sony can have some of their services temporarily taken down by Distributed Denial of Service. The response and total prevention is very difficult because of the hard task in distinguishing a legitimate user from a harmful one.

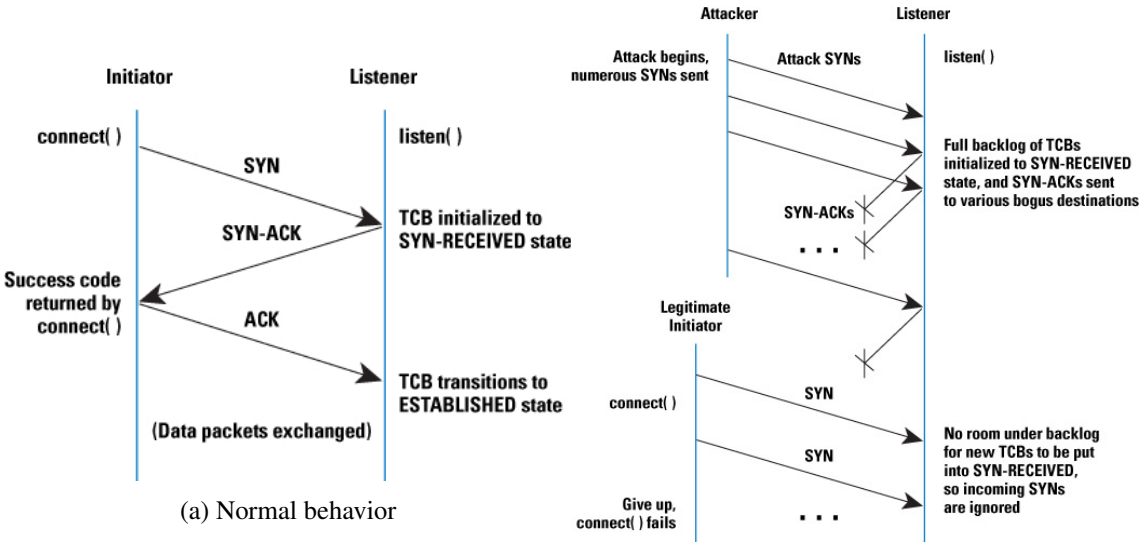
The bWAPP bee-box server was used as a target to some DoS attacks. All of these attacks were launched from one or more Linux running machines and are going to be detailed next.

One of the simplest DoS attacks to start is a ping flood. It is a layer 3 attack that works if the attacker has access to more bandwidth than the victim. This can also be upgraded to a distributed denial of service if several computers attack the same server. In this scenario the objective is just sending lots of very big Internet Control Message Protocol (ICMP) packets as fast as possible. If successful, and no protection encountered, the victim will be flooded with a vast amount of packets and forced to drop some of them, therefore the legitimate traffic the user is making can be completely dropped. Using this attack requires sudo privileges and the simple command "*sudo ping victim.com -f -s 65507*" . The f flag tells ping to send packets with a zero time interval while taking advantage of the size flag set to 65507 bytes, that coupled with the overhead of the ICMP request reaches the maximum IP packet size of 65535 bytes.

Going into OSI layer 4, the Transport Layer, it is possible to exploit the TCP handshake into causing a very dangerous DoS: the SYN flood, whose main objective is to bind all resources of the server preventing it to create TCP sessions with legitimate users. In normal operations, a TCP session is established after the three way handshake is finished. This handshake starts with the client trying to synchronize with the server, sending a TCP SYN packet. The server answers this with a TCP SYN-ACK and after that it waits for the final acknowledge from the client, a TCP ACK, shown in figure 4.3a. Only after this three steps is the session established for the TCP communication. In SYN flood DoS attack the objective is to start the handshake with the server but never actually finish it with the final acknowledgment, as is illustrated in figure 4.3b. In this way, the server remains with allocated resources for this communication that will never, theoretically, end. This attack was simulated with Low Orbit Ion Cannon, a free program with a simple user interface that can easily start SYN flooding any target [49].

At the application layer, there are several different DoS attacks that are getting more dangerous nowadays, while also not using too much bandwidth. For instance, slowloris is a python program that can exhaust the available connection of certain web servers, like Apache, without consuming much bandwidth [51]. It does this by opening several connections to the target with partial requests and holding them open for as long as possible, without actually finishing these requests. In this way the victim will allocate most of its available connections to slowloris, and when the concurrent connection pool is filled it will ultimately deny access to legitimate users.

Still in layer 7 attacks, there is a variation of slowloris that was also used in this dissertation thesis. Slowhttptest, like the name suggests, is a tool for testing your web servers against applica-



(a) Normal behavior

(b) Exploiting SYN vulnerability

Figure 4.3: TCP Handshake [50]

tion level denial of service attack [52]. This tool, obtainable through most package managers, can start an HTTP POST DoS attack. This particular attack does not make use of partial requests. Instead it sends a completely legitimate request to the target server but it changes the Content Length field of the HTTP header to represent the size of the body message that follows. The program then proceeds to send the rest of the message but at an incredible slow rate. The target, seeing a valid request, will wait for the entire message to arrive and because of that will be slowed down. With slowhttptest one can define the number of connections per second, the timeout between follow up data and the Content Length field. For instance the following terminal line will shut down any simple unprotected Apache server, like our bWAPP: "*slowhttptest -c 1000 -B -g -o my_body_stats -i 110 -r 200 -s 8192 -t FAKEVERB -u http://192.168.109.216/bWAPP/sm_dos_1.php*". From the statistics shown in figure 4.4 it only takes nine seconds for the service to be completely unavailable.

4.3 Summary

In this chapter was presented some different approaches for practicing two very important, common and recent types of cyber attacks that target a web server. If a web developer is not conscious about cyber attacks their web page can be a paradise to hackers, and exploiting SQL injections and performing DoS attacks is easy against vulnerable targets. For any web developer it is highly advised to properly escape any user input and, if the website reaches a large scale of users and accesses, one should consider to purchase some kind of DoS protection.

Detecting this attacks with our developed solution is a good step to validate its accuracy. SQL Injections can be very dangerous and go unnoticed in the middle of all capture traffic, because they only leave behind few traces in the form of HTTP requests that were made. Denial of Service

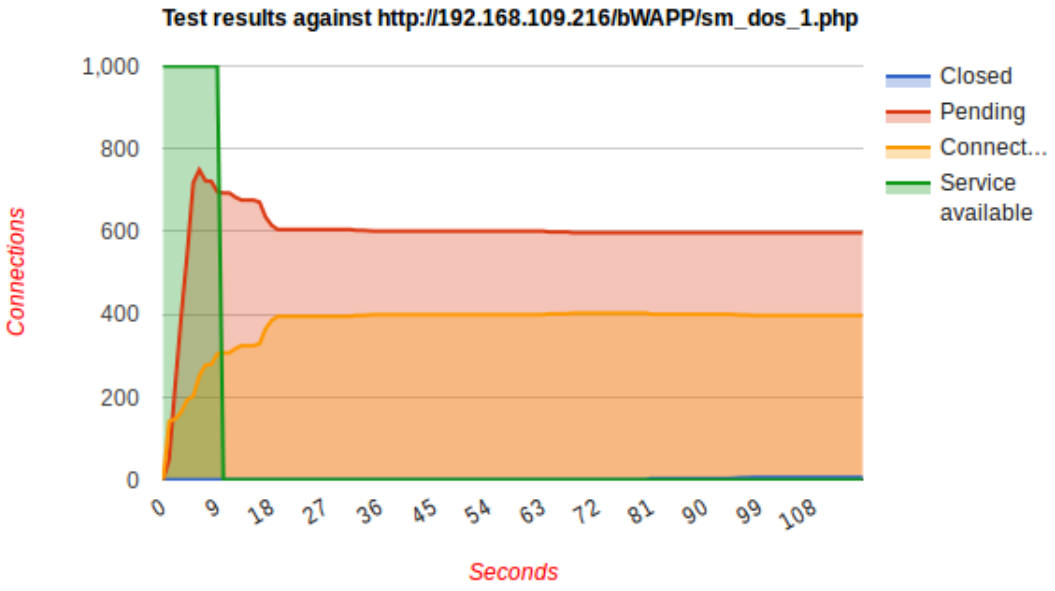


Figure 4.4: Results from slow HTTP DOS to the bWAPP server

tend to be the other way around, it is easy to notice when under attack, and usually there is large amount of traffic to be studied in the pcap capture. This are two ideal scenarios in which The Packet Hunter was tested, and for capturing the traffic that was used to create all results presented in chapter 5, all these attacks were made.

Chapter 5

Results

With all scripts written, every computer in the cluster setup with all configurations edited, and with cyber attacks ready to be made it was time to test the behavior of The Hunter. In this chapter all results obtained from different evaluations performed will be presented and explained and the methods detailed.

The testbed in which all measurements were done is given in section 5.1 and all common details to all experiments are explained on section 5.2. The developed solution was tested in distinct ways. Firstly The Packet Hunter was subjected to different kinds of cyber attacks and its performance measured and compared to the State of the Art solution, which can be seen on section 5.3. There were also made tests on the impact of different sized pcaps, section 5.4, as well as a scalability experiment of the developed platform (section 5.6), and a study of whether the division of the pcap should take place on The Packet Hunter algorithm or be part of The Capturer, presented in section 5.7.

Some conclusions to the tests done and a short round up of the information given along this entire chapter are presented in section 5.8.

5.1 Testbed

This thesis was developed in a FEUP laboratory for networking and computers. This lab, called NetLab, has three different areas. One area has six stands available for students courses and each one of these stands is composed by one Cisco 2900 series router, one Cisco Catalyst 3560 switch and four computers connected to the switch. Another area is made of ten computers in a private network that are used exclusively for making clusters with OpenStack. Finally there is an area for the development of thesis with different computers, Wi-Fi routers and Nortel switches. The network topology can be seen in figure 5.1.

OpenStack's capability of quickly setting up a computer cluster with Hadoop and a good SSH configuration was used because there was a need to start MapReduce jobs at the same time of Snort processing, therefore needing a cluster that was capable of running Snort while also having

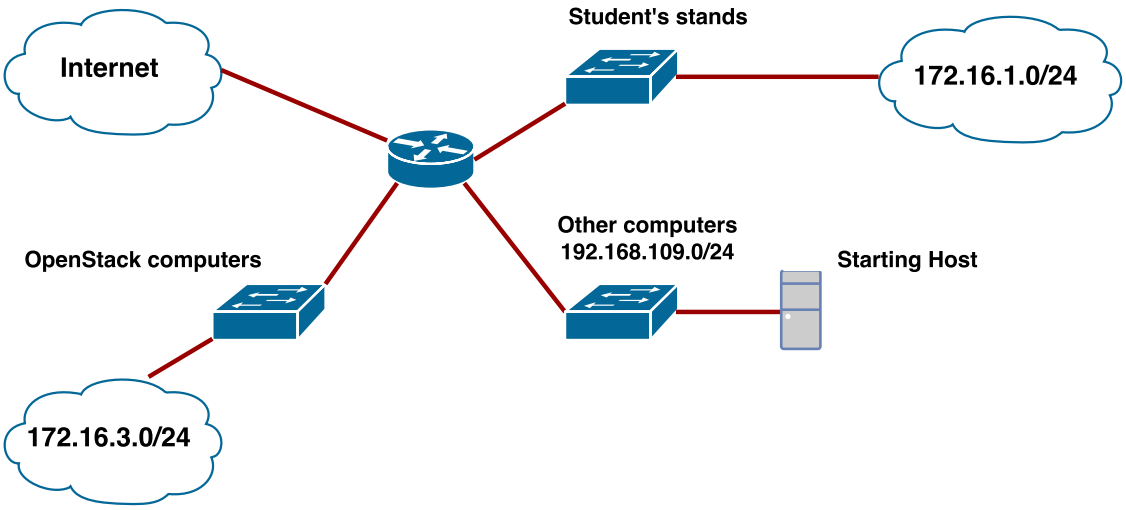


Figure 5.1: Network topology at Netlab

Hadoop configured. The physical computers used had 4GB RAM with an Intel Duo P9 Penryn. They were virtualized with OpenStack, and each instance also had 4GB RAM.

The Starting Host was always the same computer for all experiments, a Toshiba Satellite, which had 8GB RAM with an Intel i7 4510u processor with two cores and a base frequency of 2GHz.

Snort's version 3.0 is the newest stable release, at the time of writing, and was the one used. As for Snort's rules the version downloaded and configured was the 2976 snapshot, and they can be downloaded by anyone as long as an account is created on Snort's web page, which is free. Most of these rules are commented by default so users deploying the usual configuration don't get many false positives and slow performance. For our purpose all rules were used and, when testing if Snort is correctly configured, a total of 22111 rules are loaded into Snort for detection, demonstrated on figure 5.2.

```
++++++++++++++++++++++++++++++++++++++++++++++++++
Initializing rule chains...
22111 Snort rules read
    22111 detection rules
    0 decoder rules
    0 preprocessor rules
22111 Option Chains linked into 756 Chain Headers
0 Dynamic rules
++++++++++++++++++++++++++++++++++++++++++++++++++

+-------------------[Rule Port Counts]-------------
|             tcp     udp     icmp      ip
|     src    7351      76        0       0
|     dst   10432    2817        0       0
|     any    1280     202      337     192
|      nc      47      15       96       2
|     s+d      16      19        0       0
+--------------------------------------------------
```

Figure 5.2: Total detection rules loaded into Snort

5.2 Metrics to Evaluate

In this section a quick overhaul of what metrics and measurements were made to evaluate The Hunter's performance are given.

When using The Packet Hunter, the script outputs several different time measures. It shows how much time was spent dividing the pcap, using secure copy to transfer the pcap, the time it took to upload the file to HDFS (if used), the total time consumed by the Ansible playbook that distributes Snort's workload and the Snort processing time in each worker. It is important to note that the Snort processing time is actually given by Snort itself, while all other times are calculated by the python script. Also, the Ansible playbook that was timed, the one that starts Snort, includes the time for Snort loading rules, starting preprocessors and processing time, and also fetching the alerts (this will be called Ansible time further on the tests performed). When testing The Packet Hunter along side with MapReduce jobs, it was also retrieved the time spent in this step through the web interface that is made available by OpenStack.

The alerts produced in each run of The Packet Hunter are saved and their content is summarized. A comparison between the false and true positives, and accuracy, between the different tests performed is also performed.

5.3 Different Types of Traffic

In this section, the different types of attacks previously detailed on chapter 4 were put in practice. There were three very different captures performed, and each of them was processed for intrusions with The Packet Hunter. The first case scenario happened with normal traffic. In the second SQL injection exploits were practiced against the bWAPP server and the traffic captured. Finally DoS attacks were made. This resulted in three different pcaps captured in a 15 minute time.

For getting consistent results, there was a need to change the least amount of variables between tests. For that reason the cluster used always had one Master and three Workers, and The Capturer was executed with a 15 minutes cycle interval. The Starting Host was always the same, with a 1000Mb/s connection to the cluster's machines, and The Capturer was configured in local configuration for all tests. The Starting Host also had bWAPP running in VirtualBox with a bridge adapter configuration, and there was programed a simple python program to access several web pages from bWAPP, which was in turn executed in four different computers, from 172.16.1.0/24 network, to create random legitimate traffic to the web server.

A special packet capture, referenced from now on as base pcap, was also used in conjunction with the captured traffic on each test. This pcap had 1.8GB size and was captured in two hour interval, using different types of traffic like email, SSH, several web pages and forums and downloads. This base pcap served as a way to ensure the developed platform could handle more traffic than the amount that was captured in the principal tests performed.

To compare the results obtained with the developed platform, each different scenario was also tested for the state of the art solution (Snort running in only one machine) and when MapReduce jobs were added to the cluster's workload.

5.3.1 Traffic with No Attacks

The first scenario used to test The Packet Hunter involved no cyber attacks. The objective was to have a consistent set of results to use when comparing with the other obtained results from using attacks, and also to have an idea of the normal behavior of the developed platform. Three different tests were done in this scenario: one for the State of the Art solution that is Snort running in a single machine. The second test is with the normal execution of The Packet Hunter and the third corresponds to the special use of The Packet Hunter with MapReduce. For each one of this three tests 10 detection cycles were done to establish a confidence interval. Because using The Capturer and analysing different pcaps in each of the 10 detection cycles could lead to uncertain values, the same packet capture was used in all repetitions.

A traffic capture was made in 15 minutes using The Capturer in local configuration. In order to maintain consistency between this packet capture and the ones that will be made for SQL injection and DoS scenarios it was tried to have the same traffic going through the computer in all captures. This traffic can be categorized as a youtube stream, a twitch stream, facebook and linkedin opened, several email clients and a download of a pdf file and a small iso file. Constant legitimate requests to the bWAPP web server were also being done. This lead to a 891 MB pcap.

In order to bring more packets for Snort to analyze the base pcap was also used in conjunction with the captured one, so, with both pcaps a total of 2.691 GB worth of traffic were processed for intrusions in this scenario, when no attacks were done. The results obtained from all three solutions can be seen in figure 5.3, where is also represented the confidence interval based on a normal distribution with 95% confidence. The times represented are the Snort processing time and the total time to run the Ansible playbook and they are the average from all repetitions. Obviously the later is always longer than the former. Because the Snort processing time obtained in both Packet Hunter solutions are from the three Workers, there are actually 30 instances of this value for each one of these tests, so in this cases the confidence interval has this 30 values.

For a better understanding of all time values, the table 5.1 has represented the highest and lowest obtained time value for all repetitions of The Packet Hunter in its default mode, including the division and transfer time of the pcap, which were not represented on figure 5.3. One can notice that the time for dividing the capture and send it through the network are very similar and constant. Also the total time that took to process 2.691 GB pcap file with no intrusions with the developed platform was less than the 15 minute capture, which allows the system to be ready for processing the next batch of internet traffic.

When using the State of the Art solution, Snort running only in one machine, 29 different rules were triggered with a total of 1642 alerts produced. These are considered false positives as no attacks were done in this step. This is somehow to be expected as there are a large amount of rules loaded into Snort. When using The Packet Hunter in its default mode 17 different rules

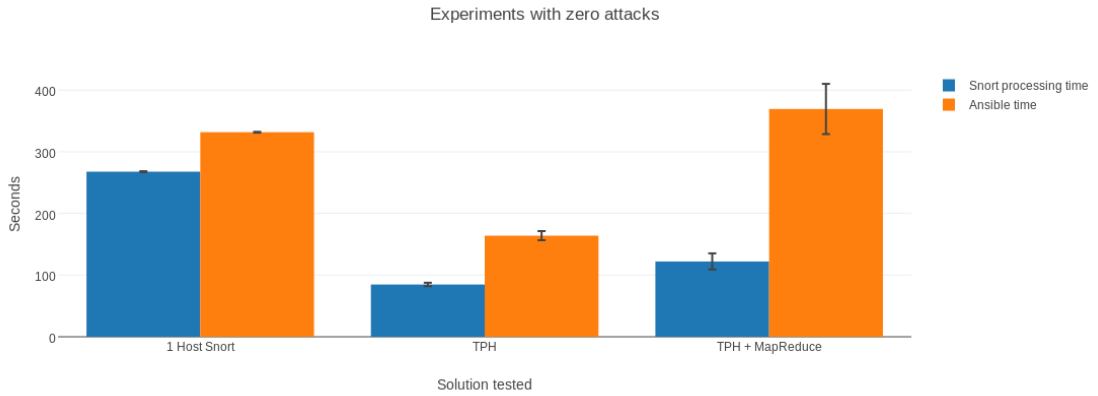


Figure 5.3: Results obtained with legitimate traffic

were detected with a total amount of 1631 alerts written. There is an obvious loss of amount of alerts produced and rules triggered when using The Packet Hunter, even though they are all false positives. This behavior is also present in all further experiments and can be caused by the separation of the original capture. This means that, when the pcap is broken in pieces there are also TCP connections and streams that get broken. Ultimately different Snort instances can not detect the attacks located at the separation borders of the original capture.

5.3.2 Traffic with SQL Injection Attacks

In this scenario the SQL Injection attacks that were detailed and experimented on section 4.1 were performed against bWAPP server and the traffic was collected with The Capturer for 15 minutes. It was expected that the results obtained here were similar to the ones in section 5.3.1, because the attacks performed did not leave a huge trace on the traffic captured. The pcap obtained had 927.5 MB and, when used with base pcap a total of 2.73 GB of traffic were processed by The Packet Hunter.

In this scenario the same tests were performed, with 10 repetitions in each one. The results obtained in these three different experiments are presented in figure 5.4. Again the same times are used: Snort's processing time and the time to run the Ansible playbook. A confidence interval of 95% for every test performed is also shown in the same figure.

Table 5.1: Time variations in The Packet Hunter for legitimate traffic

	Minimum time (s)	Maximum time (s)
Division	47.9	54.4
Secure Copy	42.7	54.7
Snort Processing	75.1	96.3
Ansible Playbook	159.8	193.2
Total Cycle Time	252.0	272.2

Figure 5.4: Results obtained with SQL Injection traffic

In table 5.2 is presented the minimum and maximum time in all recorded times obtained in the ten runs of The Packet Hunter. Just like with legitimate traffic, the values are very similar and there is no big discrepancy between them. which was expected. It is good to note that all cycle iterations finished before the 15 minute mark, so the script was ready for the next pcap.

As for the accuracy it was important to detect SQL injection attacks, in the middle of all false positive alerts, when using the proposed platform. With just Snort a total of 2343 alerts from 36 rules were written. Within this, 31 warnings about SQL injection were gatherer, including both GET and POST attempts, use of function concat and 1=1. On the other hand The Packet Hunter was able to confirm 2325 alerts from 23 different rules, however the detection of SQL injection suffered because only 15 relevant alerts were collected, which is around 50% decrease in detection rate. Even so the rules involved were the same and they allowed for a correct detection of the code injection attempts.

5.3.3 Traffic with Denial of Service Attacks

Finally it was time to test the developed solution against the types of DoS attacks that were depicted on section 4.2. Again this was done in a 15 minute interval using The Capturer, which resulted in a 1.0 GB capture, totaling 2.8 GB of traffic with the base pcap. Because this specific attack usually leaves behind many traces, it was expected that Snort took more time than the others scenarios to process, with more alerts written.

Table 5.2: Time variations in The Packet Hunter for SQL Injection traffic

	Minimum time (s)	Maximum time (s)
Division	48.2	58.2
Secure Copy	46.7	53.2
Snort Processing	76.1	94.9
Ansible Playbook	159.1	179.1
Total Cycle Time	247.7	281.3

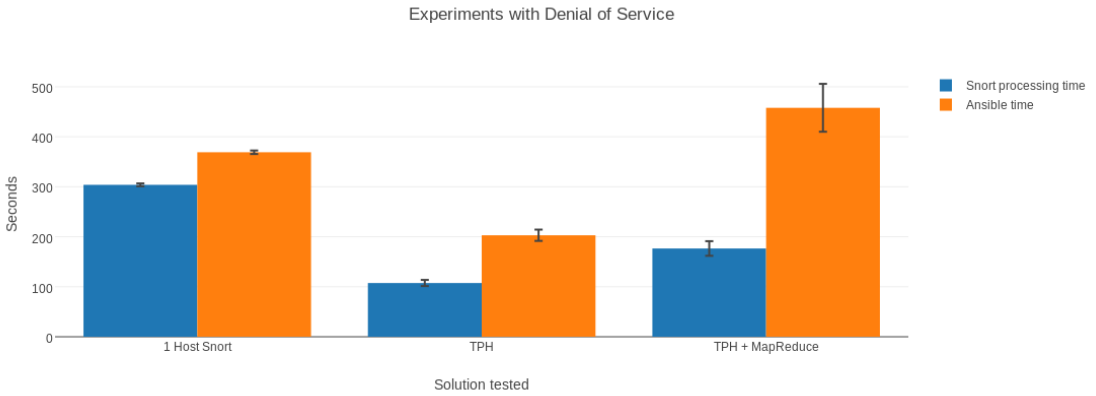


Figure 5.5: Results obtained with Denial of Service traffic

The results obtained in the three different tests performed are depicted in figure 5.5. Again, each of these was repeated 10 times and there is also shown a confidence interval of 95% for each time in the figure. In this scenario, the times obtained are generally higher than the both previous scenarios, but the Snort processing time for The Packet Hunter is still around three times less than the State of the Art solution, since three Workers are being used this behavior is mathematically correct.

In table 5.3 are written the minimum and maximum times achieved in all repetitions of The Packet Hunter in this scenario. The times for Snort processing and running Ansible are consistently higher than the times obtained with SQL and no attacks pcap, even though the capture file used here is only 70 MB bigger than the SQL one, and 109 MB than legitimate traffic pcap. The total cycle time is always less than 15 minutes, so The Packet Hunter is free for the next capture batch.

As for detection all four types of DoS depicted on section 4.2 were caught by Snort. When running it in only one machine, a total of 28903 alerts were written, for 37 different rules. With all these rules 21352 alerts were written because of irregular pings, 541 for slowloris, 115 for Low Orbit Ion Cannon and 594 for slow HTTP attack. The Packet Hunter did not underperformed here, as it actually got 21350 alerts for ping, 540 for slowloris, 106 for Low Orbit Ion Cannon and 579 for slow HTTP, with a total of 28882 alerts for 27 different rules. This represents a decrease in detection of 0.07% which can be completely ignored.

Table 5.3: Time variations in The Packet Hunter for DOS traffic

	Minimum time (s)	Maximum time (s)
Division	51.2	60.3
Secure Copy	67.3	86.5
Snort Processing	84.2	132.1
Ansible Playbook	177.3	227.2
Total Cycle Time	295.2	364.9

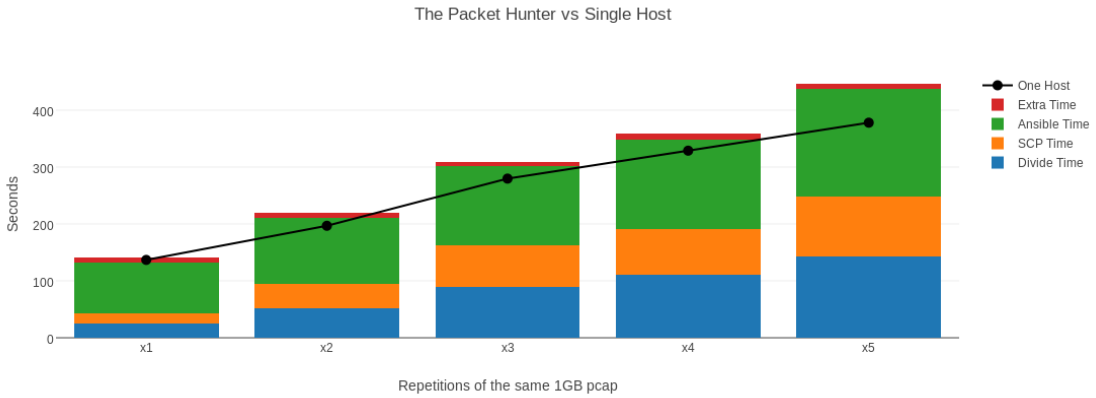


Figure 5.6: Total cycle Time with The Packet Hunter versus one host Snort implementation

5.4 Different Sized Captures

In this section is explained a different test performed on The Packet Hunter. The objective was to have an idea of the impact that different sized pcaps would have on Snort. This experiment, and the obtained results are presented next.

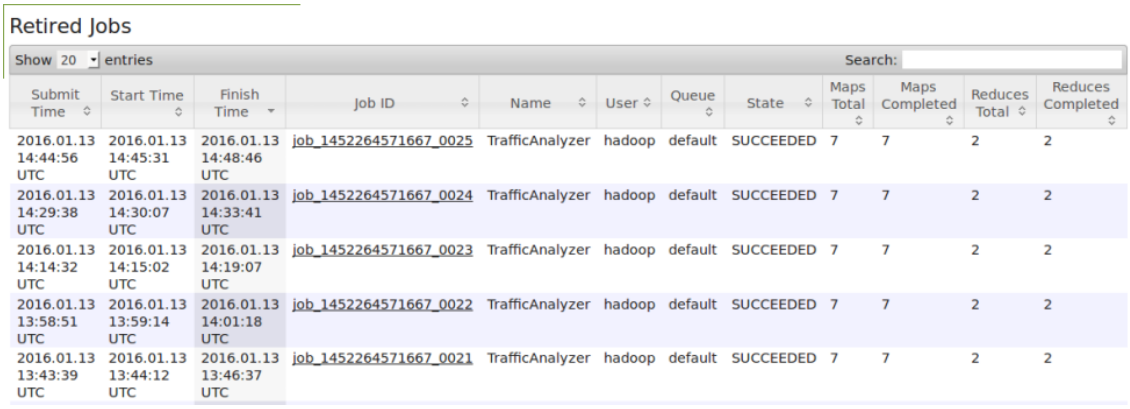
This test was done with a 1GB capture that was multiplied up to five times, and in each time the pcap (or pcaps) were processed with The Packet Hunter, saving the most important times: dividing, copying, running Ansible and total cycle time. To compare with this results the same pcap was also analyzed on a single host, using Ansible to start the detection and fetching the alerts.

The results obtained are presented in figure 5.6. The times for the different combinations of pcap when using The Packet Hunter are represented on bar graphs, while the plot line represents the total Ansible time but when dealing with a single host.

The Packet Hunter total cycle time is never faster than when running on a single host, independently of the pcap size. This is because the time spent transferring and dividing the pcap actually makes the proposed solution lose all the time that was saved by distributing the processing. As such it was also studied ways to enhance the total cycle time, including the complete removal of the division time, which is shown in section 5.7.

5.5 Hadoop and MapReduce

One important part of the tests performed on The Packet Hunter involved using MapReduce jobs on the workers that were, at the same moment, processing packets with Snort. The objective was to access if both tasks could be done in parallel at the same time without degrading Snort's performance. The function used for the MapReduce job was a packet classification, developed by students at FEUP.



Retired Jobs

Show 20 entries Search:

Submit Time	Start Time	Finish Time	Job ID	Name	User	Queue	State	Maps Total	Maps Completed	Reduces Total	Reduces Completed
2016.01.13 14:44:56 UTC	2016.01.13 14:45:31 UTC	2016.01.13 14:48:46 UTC	job_1452264571667_0025	TrafficAnalyzer	hadoop	default	SUCCEEDED	7	7	2	2
2016.01.13 14:29:38 UTC	2016.01.13 14:30:07 UTC	2016.01.13 14:33:41 UTC	job_1452264571667_0024	TrafficAnalyzer	hadoop	default	SUCCEEDED	7	7	2	2
2016.01.13 14:14:32 UTC	2016.01.13 14:15:02 UTC	2016.01.13 14:19:07 UTC	job_1452264571667_0023	TrafficAnalyzer	hadoop	default	SUCCEEDED	7	7	2	2
2016.01.13 13:58:51 UTC	2016.01.13 13:59:14 UTC	2016.01.13 14:01:18 UTC	job_1452264571667_0022	TrafficAnalyzer	hadoop	default	SUCCEEDED	7	7	2	2
2016.01.13 13:43:39 UTC	2016.01.13 13:44:12 UTC	2016.01.13 13:46:37 UTC	job_1452264571667_0021	TrafficAnalyzer	hadoop	default	SUCCEEDED	7	7	2	2

Figure 5.7: Web interface results of MapReduce jobs

The results obtained were presented in figures 5.3, 5.4 and 5.5. On those images, what is represented as Ansible time is the total of starting Snort (including loading all rules and configuring preprocessors), processing the traffic capture and fetching the alerts produced. However when using MapReduce, because both tasks are being done in parallel, the total time of running the MapReduce jobs was also included in the Ansible time represented in the respective use case scenarios.

When comparing the results obtained with using the traffic that has SQL injection attacks (5.4) it is possible to observe that, even with the same exact pcap, the MapReduce solution took 33.1% (28.1 seconds) more time in the Snort's processing and 123.9% (203.4 seconds) plus total Ansible time. The total time used by MapReduce, since the job was submitted till it ended, was saved through the web interface provided by OpenStack. Five of this jobs can be seen in figure 5.7. Of the ten jobs that were started, an average of 213.2 $\pm$25.3 seconds were used by MapReduce for concluding each one. Even though this time is less than the total Ansible time (367.5 $\pm$35.1 seconds) it is possible to affirm that Snort's performance is degraded by other processing, but not to an extent of being unbearable, since The Packet Hunter still processed everything before 15 minutes had passed, and another capture arrived.

5.6 Scalability Experiment

In this experiment, two different sized clusters were used to analyse the exact same pcap and the results obtained in the Ansible time and each worker processing time was compared in both ways.

In table 5.4 is presented the results obtained when processing a 2.0GB pcap, which contained some DoS attacks, with two different sized clusters: one with four workers and the other with five. The same processing was repeated ten times for each cluster and the results obtained are shown in seconds with a 95% confidence interval.

It is easy to conclude that more computers are equal to faster processing. In this case there was achieved an average of 21.6% gain in time in each worker and a 13.1% gain for Ansible time.

Table 5.4: Times for different sized clusters when processing the same pcap.

	Workers Processing (s)	Total Ansible time (s)
4 Workers	67.0 ±1.3	143.4 ±1.5
5 Workers	52.5 ±1.1	124.6 ±1.0

5.7 Studying the Division Step of The Packet Capture

In The Packet Hunter algorithm, the first thing that happens, as soon as the detection cycle starts, is the division of the pcap that was previously captured in equal smaller parts for the computer workers to process. This is a time consuming process, directly proportional to the size of the pcap to be divided, that makes the total detection cycle time of this proposed solution slightly worse when comparing to the State of the Art solution. This step was done with the assumption that it was better to distribute the same size of pcaps to Snort for an even processing with the Workers.

An alternative to this process would be to have The Capturer delivering not one, but a number of smaller pcaps to The Packet Hunter (according to the number of workers in the cluster), meaning that there will be no time spent dividing the pcap by the later script. For instance, if there are three workers and a 15 minute capture is made, then The Capturer would make three pcaps, one for each 5 minute set. The problem with this alternative is that the three pcaps would only have a similar size between them if the network traffic remains constant. If a peak occurs, one of the files could be substantially bigger than the rest, and can delay the distributed processing.

To study this effect both scripts were changed to represent this alternative. In theory eliminating the division time would produce less time for the total detection cycle if the captured traffic is constant, so only the worst case scenario was practiced. A cluster with three workers was used and The Capturer was set, once again, to a 15 minute local configuration. To simulate a peak of internet traffic, a ping flood attack was performed on the first five minutes of capture leading to three consecutive pcaps: the first one of 1.4GB, the second with 258MB and the third with 416MB. Then the same pcaps were used but with the division step and the times obtained recorded.

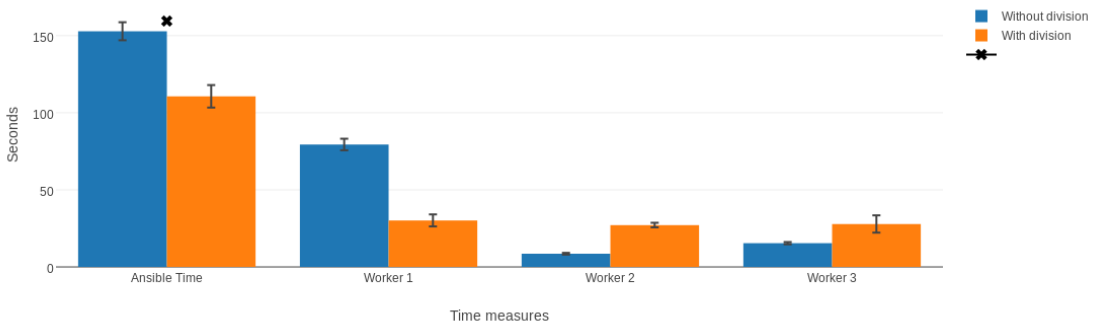


Figure 5.8: Comparing times when using, or not, the dividing step

The results obtained when using division on The Packet Hunter, and when letting the pcap be already divided by The Capturer are shown in figure 5.8. The times represented are, for both alternatives, the total Ansible time and the Snort processing time at each Worker. Because the captures in both tested ways are the same the transfer time is not represented, as it only depends on the network. Ten repetitions were made and a confidence interval of 95% is also represented.

It is possible to notice that the columns on the right of each measured time are constant while the new alternative has a time spike in worker 1. This is to be expected because the pcaps were evenly distributed which lead to an approximate processing time by Snort on each worker. On the other hand, when the division step is not performed by The Packet Hunter, there is indeed a spike of processing on Worker 1 which had to analyze the pcap with the huge DoS traffic, which lead to a rise on the total Ansible time. In order for this new alternative to be better, the time spent dividing the pcap, on the alternative that did it, needed to be greater than the difference between both Ansible times, so that the time that was gained in one step would not be lost in another. The division time recorded was 49.5 $\pm$0.7 seconds and, when added to the correspondent Ansible time it actually surpasses the alternative. This operation is depicted on the cross in the figure.

From this results, which were made in the worst possible scenario, it is right to conclude that The Packet Hunter should not divide the capture like it was doing. Instead it should be up to The Capturer to do this by dividing the total time of capture by the number of computer workers in the cluster.

5.8 Discussion

In this chapter the testbed in which all experiments were done is presented, the tests that The Packet Hunter was subjected to were detailed and the final results were given. In summary, The Packet Hunter was tested for its ability to detect intrusions while the performance was being measured. Several different packet captures were used: one with legitimate traffic, another with SQL injection attacks and the other under DoS. The results obtained were compared with the State of the Art solution (Snort running on a single host) and for the special case of using MapReduce jobs along with Snort intrusion detection.

In all scenarios it is possible to see that the developed platform was around three times faster than the State of the Art solution when it came to Snort's processing time. Because the cluster used for distributing workload also had three computer Workers, this is an acceptable and predictable result. On the other hand it was not predicted that the use of MapReduce jobs in conjunction with Snort's processing was such a detriment to the proposed tool. In all cases the processing time suffered consistent delays, and the total Ansible time had it even worse, mostly because it had to wait for the end of the execution of the jobs. Even with all this delay the program would always be done under the 15 minute mark.

The intrusion detection when using The Packet Hunter was generally worse than a single host Snort. When dealing with DoS attacks the alerts produced were almost the same, because they leave such high evidence in the traffic. Unfortunately SQL injection attempts were not as well

detected. This may happen due to the original packet capture being divided, and as such the TCP/IP flows are segmented to each smaller pcap is lost to the distribution of Snort, while the State of the Art solution, that gets the full pcap, can completely analyze the stream. By looking at the number of alerts that are lost, instead of the proportion, there are 16 lost alerts with SQL traffic and 21 with DoS, which is very similar. Even so all cyber attacks practiced were detected by the proposed solution.

Another test performed involved studying the impact bigger pcaps would have on the proposed platform, and the total detection cycle time was charted against the single host time for running Snort. This resulted in an almost linear graph where it could be observed that the total cycle time for The Packet Hunter is almost the same as the processing of Snort in a single host of the same pcap, even when including the time to divide the pcap and transfer it to the cluster, validating the hypotheses that the developed solution can keep up with the State of the Art one, as far as time consumed is concerned.

Will all the tests performed, it can be concluded that the proposed platform can be used in any environment, with different sized captures and it can detect several cyber attacks. The Hunter has different scripts that allow this tool to be very easily configured in a previously deployed cluster, with integrated Hadoop support. It is also completely automated upon start, and scalable to any sized cluster.

Chapter 6

Conclusion

This dissertation theme had the objective of studying possible enhancements for intrusion detection, a field that needs all the help it can get to fight the ever-growing cyber crime. It was chosen to work with Snort, one of the most used IDS based on predefined rules, and it was tackled its slow performance when using a large amount of them. By using the distributed computing capabilities of Hadoop with rule-based IDS we aim to distribute the detection to a cluster and addresses the low processing speed that occurs when too much packets are analyzed and to many rules loaded, while keeping the detection ratio with high standards.

In the time spent working for this dissertation thesis, some problems arose and difficulties where encountered. Sometimes dead ends were reached and new ideas born from that. In the end The Hunter was developed, a completely automated and scalable platform for distributing Snort's processing workload to a computer cluster. There were implemented functions for using a MapReduce job along with Snort, for further integration with the Hadoop framework.

6.1 Contributions

The major contribution of this thesis was The Hunter, a tool which contains a number of scripts that, in a completely automated and scalable way, distribute a intrusion detection program trough a computer cluster. This tool has two very important scripts: The Packet Hunter, detailed on section 3.3 and The Capturer, further explained on section 3.2.

This proposed platform was exhaustively tested in different ways. The intrusion detection capability of The Packet Hunter had to be asserted and compared against a Snort implementation in only one host. With this thesis we were able to prove that the developed tool could detect every single attack made, although with an inferior detection rate. It could also handle different sized pcaps in the same time proportions as the State of the Art counter part.

A study of Snort's performance, when dealing with different types of traffic and working together with MapReduce and the Hadoop framework is also a very important contribution of this dissertation.

6.2 Future Work

To make The Packet Hunter more effective, and unlock its true potential, there are a number of improvements that could be made, which are:

- One of the steps in the algorithm of The Packet Hunter is the transfer of the pcaps to all computers in the cluster. This is done with secure copy, which creates a SSH tunnel between both ends and encrypts the connection. Even though transfer rates of 70 to 80 MB/s could be achieved, other solutions could be tested for even less transfer time.
- To achieve a more diversified IDS, the MapReduce job used in conjunction with The Packet Hunter detection cycle could be change to an anomaly-based algorithm for intrusion detection, like outliers for instance. If this is implemented both rules and anomaly based intrusion is being performed.
- If there is a spike in network traffic, there is a chance that The Packet Hunter will not process everything before the next packet capture arrives, since it is always working in cycles. For that reason, making the algorithm adaptive to the time used will bring robustness to the proposed platform.
- General improvements to The Hunter tool, such as a graphical interface for a more user friendly environment and other new features.

Bibliography

[1] *Gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015*, http://www.gartner.com/newsroom/id/3165317, [Online; accessed 22-December-2015].

[2] K. Lab and Interpol, "Mobile cyber threats", 2014.

[3] K. L. G. Research and A. Team, *Kaspersky security bulletin 2013*, http://media.kaspersky.com/pdf/KSB_2013_EN.pdf, [Online; accessed 20-September-2015], 2013.

[4] *Big data at the speed of business*, http://www-01.ibm.com/software/data/bigdata/industry.html, [Online; accessed 18-September-2015].

[5] *What is apache hadoop?*, http://hadoop.apache.org/, [Online; accessed 22-September-2015].

[6] M. Kumar and M. Hanumanthappa, "Scalable intrusion detection systems log analysis using cloud computing infrastructure", in *Computational Intelligence and Computing Research (ICCIC), 2013 IEEE International Conference on*, IEEE, 2013, pp. 1–4.

[7] https://sourceforge.net/projects/libpcap/, [Online; accessed 11-December-2015].

[8] *Tcpdump documentation*, http://www.tcpdump.org, [Online; accessed 18-October-2015].

[9] *About wireshark*, https://www.wireshark.org/about.html, [Online; accessed 17-October-2015].

[10] eWEEK Labs, *The most important open-source apps of all time*, http://www.eweek.com/c/a/Linux-and-Open-Source/The-Most-Important-OpenSource-Apps-of-All-Time/5, [Online; accessed 17-October-2015].

[11] WinPcap, *The industry-standard windows packet capture library*, http://www.winpcap.org/default.htm, [Online; accessed 21-December-2015].

[12] NirSoft, http://www.nirsoft.net/utils/smsniff.html, [Online; accessed 23-December-2015].

[13] https://www.nagios.org/, [Online; accessed 12-December-2015].

[14] http://www.zenoss.com/, [Online; accessed 12-October-2015].

[15] http://www.ntop.org/, [Online; accessed 12-December-2015].

[16] U. K. D. in Databases, *Kdd cup 1999 data*, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, [Online; accessed 10-October-2015], 2013.

[17] N. G. Relan and D. R. Patil, "Implementation of network intrusion detection system using variant of decision tree algorithm", in *Nascent Technologies in the Engineering Field (ICNTE), 2015 International Conference on*, IEEE, 2015, pp. 1–5.

[18] C. A. Kumar *et al.*, "Intrusion detection model using fusion of PCA and optimized SVM", in *Contemporary Computing and Informatics (IC3I), 2014 International Conference on*, IEEE, 2014, pp. 879–884.

[19] Y. Guang and N. Min, "Anomaly intrusion detection based on wavelet kernel ls-svm", in *Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on*, IEEE, 2013, pp. 434–437.

[20] *Elki: environment for developing kdd-applications supported by index-structures*, http://elki.dbs.ifi.lmu.de/, [Online; accessed 22-June-2015].

[21] F. Y. Ke, F. Yan, and Z. J. Lin, "Research of outlier mining based adaptive intrusion detection techniques", in *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, IEEE, 2010, pp. 552–555.

[22] Z. Mingqiang, H. Hui, and W. Qian, "A graph-based clustering algorithm for anomaly intrusion detection", in *Computer Science & Education (ICCSE), 2012 7th International Conference on*, IEEE, 2012, pp. 1311–1314.

[23] E. Raftopoulos and X. Dimitropoulos, "Ids alert correlation in the wild with edge", *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 10, pp. 1933–1946, 2014.

[24] S. Roschke, F. Cheng, and C. Meinel, "A flexible and efficient alert correlation platform for distributed ids", in *Network and System Security (NSS), 2010 4th international conference on*, IEEE, 2010, pp. 24–31.

[25] S. Chadli, M. Saber, M. Emharraf, and A. Ziyyat, "A new model of ids architecture based on multi-agent systems for manet", in *Complex Systems (WCCS), 2014 Second World Conference on*, IEEE, 2014, pp. 252–258.

[26] O. Rodas, G. Morales, and J. Alvarez, "A reliable and scalable classification-based hybrid ips", in *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, IEEE, 2015, pp. 599–604.

[27] SNORT, https://www.snort.org/, [Online; accessed 17-October-2015].

[28] J. Xu, J. Zhang, T. Gadipalli, X. Yaun, and H. Yu, "Learning snort rules by capturing intrusions in live network traffic replay", *Proc. 15th Colloquim for Information Systems Security Education*, pp. 145–150, 2011.

[29] M. Roesch *et al.*, "Snort: lightweight intrusion detection for networks.", in *LISA*, vol. 99, 1999, pp. 229–238.

[30] H. Alnabulsi, M. R. Islam, and Q. Mamun, "Detecting sql injection attacks using snort ids", in *Computer Science and Engineering (APWC on CSE), 2014 Asia-Pacific World Congress on*, IEEE, 2014, pp. 1–7.

[31] A. Hashim, "The integration of snort with k-means clustering algorithm to detect new attack", *University of Technology MARA, Malaysia*, 2011.

[32] https://www.bro.org/, [Online; accessed 1-December-2015].

[33] V. Paxson, "Bro: a system for detecting network intruders in real-time", *Computer networks*, vol. 31, no. 23, pp. 2435–2463, 1999.

[34] *Openstack open source cloud computing software*, http://bigdataanalyticsnews.com/hadoop-yarn-adds-application-threads-big-data-users/, [Online; accessed 28-September-2015].

[35] J. Li, X. Lin, X. Cui, and Y. Ye, "Improving the shuffle of hadoop mapreduce", in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, IEEE, vol. 1, 2013, pp. 266–273.

[36] Y. Tang, E. Abdulhay, A. Fan, S. Su, and K. Gebreselassie, "Multi-file queries performance improvement through data placement in hadoop", in *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, IEEE, 2012, pp. 986–991.

[37] R. Morla, P. Gonçalves, and J. Barbosa, "A scheduler for cloud bursting of map-intensive traffic analysis jobs", *Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015): Krakow, Poland.*, pp. 11–21, 2015.

[38] M. Gavahane, D. Sequeira, A. Pandey, and A. Shetty, "Anti-phishing using hadoop-framework",

[39] S. Kulkarni, M. Mutalik, P. Kulkarni, and T. Gupta, "Honeydoop-a system for on-demand virtual high interaction honeypots", in *Internet Technology And Secured Transactions, 2012 International Conference for*, IEEE, 2012, pp. 743–747.

[40] J. Therdphapiyanak and K. Piromsopa, "An analysis of suitable parameters for efficiently applying k-means clustering to large tcpdump data set using hadoop framework", in *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2013 10th International Conference on*, IEEE, 2013, pp. 1–6.

[41] https://github.com/packetloop/packetpig, [Online; accessed 10-January-2016].

[42] M. Baker, D. Turnbull, and G. Kaszuba, *Finding needles in haystacks (the size of countries)*, http://docs.huihoo.com/blackhat/europe-2012/bh-eu-12-Baker-Needles_Haystacks-WP.pdf, 2012.

[43] https://www.ansible.com/, [Online; accessed 12-December-2015].

[44] SNORT, https://www.snort.org/faq/why-are-rules-commented-out-by-default, [Online; accessed 20-January-2016].

[45] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop", *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 1, pp. 5–13, 2013.

[46] OWASP, "Owasp top 10 - 2013", *The Ten Most Critical Web Application Security Risks*, 2013.

[47] http://sqlmap.org/, [Online; accessed 2-October-2015].

[48] http://www.itsecgames.com/, [Online; accessed 27-September-2015].

[49] http://sourceforge.net/projects/loic/, [Online; accessed 26-September-2015].

[50] CISCO, *Defenses against tcp syn flooding attacks*, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_9-4/syn_flooding_attacks.html, [Online; accessed 2-December-2015].

[51] https://github.com/llaera/slowloris.pl, [Online; accessed 25-September-2015].

[52] https://github.com/shekyan/slowhttptest, [Online; accessed 10-December-2015].